5-2019

Twitter Sentiment Analysis on Leicester City's Phenomenal 2015/16 EPL Title Winning Season

Eyosyas Dagnachew

Twitter Sentiment Analysis on Leicester City's Phenomenal 2015/16 EPL Title Winning Season

Eyosyas Dagnachew

Woodbridge, Virginia

A thesis submitted in partial fulfilment of the requirements for the degree of

Bachelor of Science in Computer Science from William & Mary

Department of Computer Science

Advisor: James Deverick

William & Mary

May 2019

Twitter Sentiment Analysis
on Leicester City's Phenomenal
2015/16 EPL Title Winning Season

A thesis submitted in partial fulfillment of the requirements
for the degree of Bachelor of Science in Computer Science from
William & Mary

by

Eyosyas Wolde Dagnachew

Accepted for Honors

James Deverick, Director

Robert Michael Lewis

Daniel Parker

Williamsburg, VA
May 3, 2019

Acknowledgement

I would like to first thank my advisor James Deverick. This project would not have been possible without the motivation, guidance, and support he provided me throughout my senior year at the College of William & Mary. I would also like to thank Dr. Robert Lewis for giving me feedback on my project. I would like to thank my honors committee consisting of Dr. Robert Lewis, Dr. Daniel Parker, and James Deverick for agreeing to serve on my committee and making this thesis possible. I would like to thank the computer science community, especially Scikit-Learn[1] and Stack Overflow[2], for providing resources and guidance to help me complete this project. I would like to thank my friends for providing support and motivation. I would especially like to thank Jennifer Traver for helping me label my training data and Dmytro Shmagin for providing encouragement and feedback on my project. Finally, I would like to thank my family, especially my parents, for providing their unwavering support and motivation throughout this project and throughout all of my other academic endeavors.

[1] https://scikit-learn.org/stable/
[2] https://stackoverflow.com/

Abstract

Microblogging has become one of the most useful tools for sharing everyday life events and news, especially popular sporting events, and for expressing opinions about those events. The English Premier League (EPL), the most popular professional soccer league in the world, is talked about on Twitter every day, and the 2015/16 season, whose title underdogs Leicester City managed to win, was one for the history books to remember. As Twitter posts are short and constantly being generated, they are a great source for providing public sentiment towards events that occurred throughout the 2015/16 EPL season. In this project, we examine the effectiveness of machine learning and text sentiment analysis on classifying the sentiment of tweets about Leicester City. We accomplish this by collecting tweets containing the words “Leicester City” using the python library GetOldTweets3[3]; manually labelling those tweets as positive, negative, or neutral; and training an SVM classifier to classify tweets about Leicester City from the 2015/16 season. Our model achieved an F1-score of 0.76. We use the sentiments returned from the classifier to find correlations between real-life events and sentiment changes throughout the whole season and during individual games. From our analysis, we discovered an increase in tweets about Leicester City but a sentiment change from positive to negative as the season progressed. We also observed a wide range of changes in sentiment during a single match involving Leicester City due to real-life events as well as other factors which we discuss in detail.

[3] https://pypi.org/project/GetOldTweets3/

Table of Contents

1. Introduction

1.1. Motivation for Thesis

As of November 2018, around 6,000 tweets are sent on average every second [1], which corresponds to around 350,000 tweets per minute, 500 million tweets per day, and 200 billion tweets per year. Popular events such as elections, tragedies, discoveries, and sporting events lead to increases in these numbers. For example, the 2016 US Presidential Election, in which Trump claimed victory, saw the most-tweeted election day ever with 75+ million tweets around the world talking about the election. The 2014 FIFA World Cup saw 672 million tweets about the competition sent from the start until the end of the tournament [2]. These tweets contain important information that could be used for numerous reasons, some of which we will discuss in later sections. However, the substantial quantity of the tweets means that it is virtually impossible to manually analyze these tweets.

Machine learning, which has seen a significant rise in popularity in the past several decades, is being implemented to read and analyze these large amounts of data. Furthermore, sentiment analysis is being used to understand the opinions of a population. The rise of Twitter, especially, has enabled soccer fans to constantly post updates about games or soccer related events. Fans use Twitter to express their feelings about goals scored, players transferred to different teams, and bizarre events on and off the field. In this project, we are mainly interested with the sentiments that fans on social media express during soccer games. We want to analyze how and why these sentiments change due to events that occur during matches, and we want to discuss how these results could be utilized. Furthermore, we want to perform a case study on Leicester City and their title winning 2015/16 EPL season. That season came as a surprise to the world of soccer, and we want to analyze the feelings that people expressed towards Leicester City and how those feelings changed throughout the season and during important games.

1.2. Outline

This work is structured as follows: we first discuss the interesting 2015/16 EPL season, why Leicester City's victory was such a surprise to the world, and why this could be worth analyzing.

In the next section, we review some related works involving the use of text classification on topics associated with soccer. In Section 4, we provide a quick introduction to the concepts and terminologies of machine learning and sentiment analysis, and we talk about some of the known applications of sentiment analysis. In Section 5, we briefly introduce Support Vector Machines, list known use cases for SVMs and discuss why we chose to use a linear SVM for this project, and finally compare the pros and cons of SVMs. In the next section, we lay out the details of our methodology and implementation for this project. This includes the specifics on the data set, preprocessing, and training. The next section discusses the results of our final model. We then present some analysis work we did with our classification results. Finally, we summarize our contributions in the conclusion section and list out ideas for future work.

2. Leicester City Football Club

In May 2014, Leicester City F.C. won the EFL Championship title [3], the second-highest division in the English soccer league system after the English Premier League, and they were promoted into the EPL. Despite some highlights, they had an awful run of results in the first 29 games and were in last (20th) place during the 2014/15 EPL season. Leicester miraculously won seven of their last nine games for that season and escaped relegation (bottom 3 of 20 teams) back to the EFL Championship, finishing 14th in the EPL. During the summer of 2015, Leicester surprisingly appointed Claudio Ranieri as their new coach [4]. Ranieri announced that the goal for the club in the upcoming season was to reach 40 points, which is regarded as the minimum required to avoid relegation. The rest of the season for Leicester City will forever be in the soccer history books.

For casual soccer fans, it could be hard to understand how ridiculously unlikely it was for Leicester City to win the title. At the start of the 2015/16 EPL season, Leicester were 5000-1 amongst bookmakers to win the title. To put this in perspective with other sports, the so-called “Miracle Mets” were only 100-1 when they won the 1969 World Series and Buster Douglas was a 42-1 underdog when he upset Mike Tyson to win the heavyweight champion in 1990. From the perspective of non-sporting and quite ridiculous events, the chances of Elvis Presley being found alive and Barack Obama playing cricket for England both had odds of 5000-1 [5], the same as Leicester City winning the league in 2015.

Leicester started the season strong with a 4-2 win over Sunderland on the first day. They stayed near the top of the table until their first loss of the season on September 26, 2015, a 5-2 crushing home defeat against Arsenal. Although their offense was firing, their defense was inconsistent and was struggling to keep out goals. Eventually, they improved and Jamie Vardy, Leicester’s main striker, equaled then broke the current standing record for goals scored in consecutive games in the Premier League (11), which was held at the time by legendary Manchester United striker Ruud van Nistelrooy. Leicester spent Christmas Day at the top of the table and had the world starting to ask questions about their legitimate contest for the title. They eventually slipped up again and lost to Liverpool 1-0 but quickly went back on top in January.

Arsenal yet again broke Leicester hearts on February 14, 2016 with a last-minute winner, but the result did not remove Leicester from the top of the table. In fact, that would be the last (and only third) game Leicester lost for the rest of the season. Although they lost points from games ending in draws, Leicester eventually won the Premier League title by going 7 points ahead with two games to spare on May 2, 2016 when their closest rival on the table, Tottenham Hotspurs, drew 2-2 against Chelsea and were no longer able to overtake Leicester.

Figure 1: Leicester City raising the EPL trophy on May 7, 2016 [6]

That season, Leicester broke numerous records. Alongside Jamie Vardy, Riyad Mahrez also broke a new record when he became the first Algerian player to receive the PFA Players' Player of the Year award [7], an annual award given to the player who is considered to have been the best player of the year in English soccer. Before Leicester City won, only five other teams (Manchester United, Arsenal, Chelsea, and Manchester City, and Blackburn Rovers) had won the EPL title since its founding in 1992. Every year, the title-winning team had finished in the top three spots during the previous year. Leicester won the league after barely escaping relegation and finishing 14th in the previous year.

BBC Sports described Leicester's 2015/16 season as "one of the greatest sporting stories of all time." [8] People all over the world were talking about Leicester City on all kinds of platforms, especially on social media and especially on Twitter. One way to find out what everyone was talking

about and how they felt about the results is to utilize sentiment analysis, specifically sentiment polarity classification which involves the labeling of opinionated text and categorizing it into positive, negative, and neutral classes. Through this process, we hope to see how people's thoughts about Leicester City changed as the season progressed and discover patterns based on opinionated tweets.

3. Related Work

Sentiment analysis has previously been used on sporting related research for numerous tasks. A research work similar to our project was done by Barnaghi et al. [9] who used Logistic Regression Classification (LRC) to classify sentiments of tweets posted during a major event in the 2014 World Cup, when Uruguayan striker Luis Suarez purposefully bit Italian defender Giorgio Chiellini. Their results showed that people express negative sentiments towards unethical behavior. Similarly, Yu and Wang [10] used sentiment analysis to examine U.S. soccer fans' emotional responses, particularly when goals were scored, from tweets collected in real-time during five 2014 FIFA World Cup games. Three of the five games involved the U.S. national soccer team. They found that the most common negative sentiments during U.S. games were fear and anger, which increased when the opponent scored and decreased when the U.S. team scored. Anticipation and joy also appeared to be consistent with the results of those games. In the other two games in which the U.S. team was not involved, anticipation and joy were more present than fear and anger. The responses to goals scored were also unclear. Overall, Yu and Wang's work showed that sports fans utilize Twitter to reveal their emotions and that emotional changes were consistent with the events of the games when the fanship was valid. Gratch et al. [11] also used sentiment analysis of tweets from the 2014 World Cup. They concluded that contrary to assumptions in sports economics, excitement actually relates to expressions of negative emotion.

Slightly deviating from sentiment analysis, Van Oorschot et al. [12] conducted an interesting project in which they scraped and analyzed tweets about 61 Dutch premier league soccer games to extract the minutes in which an event occurred, classify the event type, and assign events to either the home or away team. Jai-Andaloussi et al. [13] conducted a similar research but instead used sentiment analysis of tweets sent during soccer games to detect and predict the team supported by each fan, the teams and players involved in each tweet, and the details associated with each event. They used this information to create a summary of soccer matches.

Furthermore, sentiment analysis has also been used to build a system to predict match outcomes and use as a wagering decision system [14].

4. Machine Learning and Sentiment Analysis

4.1. A Brief Introduction to Machine Learning

Before we discuss the details of sentiment analysis, it is vital to have an understanding of machine learning. Machine learning is an “application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.” [15] Machine learning problems can be categorized into various branches. For this project, we are primarily interested in Supervised Text Classification. The “Supervised” part of that term refers to the learning method that is usually used for classification problems, such as the one in this project where we manually label (in other words “supervise”) our data before feeding it into our model. The other learning methods include semi-supervised, unsupervised, and reinforcement learning. The “Text” part of the above term just means that we will be dealing with textual data. Other forms of data machine learning can be used for include imagery and audio data. The “Classification” part refers to the type of problem we are trying to solve, i.e. we are attempting to classify a text as containing positive, negative, or neutral sentiments. Other examples of classification problems include face detection, email spam filtering, medical diagnosis, and weather prediction [16]. Machine learning can also be used to solve regression problems, problems in which the output variable is a real or continuous value such as salary, age, or weight [17].

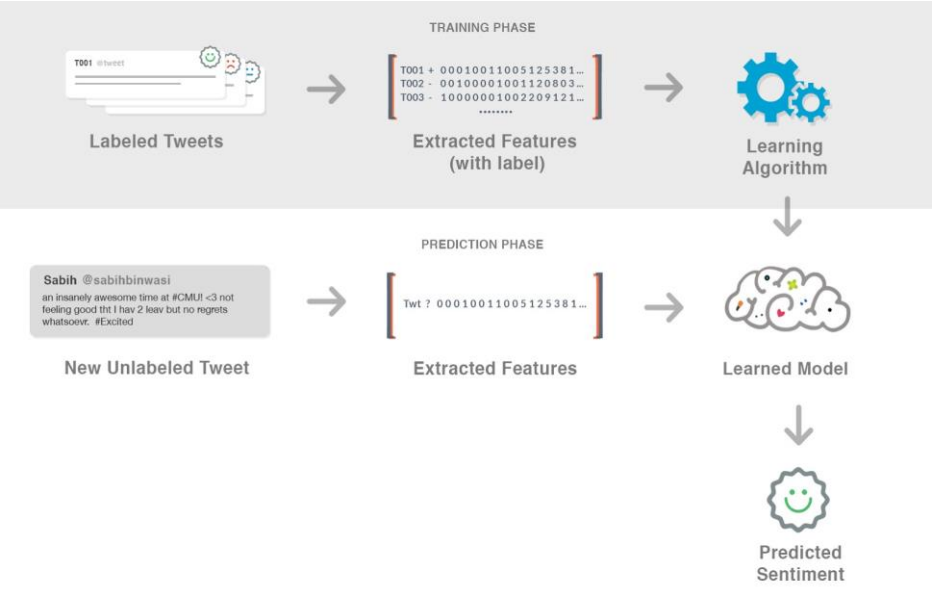

Figure 2: Overview of Supervised Sentiment Classification of Tweets [18]

The approach for Supervised Text Classification, visualized in Figure 2, is as follows:

1. Compile a data set of labelled text to use for training.
2. Use a feature extractor to convert texts into a numerical representation that the model can read.
3. Feed the feature vectors into the classification algorithm, which will attempt to find relations between each feature (value) in a vector and the labelled sentiment.
4. Store the learned model and use it to predict new data.

4.2. A Brief Introduction to Sentiment Analysis

Sentiment analysis, which is also referred to as opinion mining, is defined as the process of automatically identifying and extracting polarity, emotions, opinions, attitudes, and views from text sources through Natural Language Processing (NLP). Although words like opinion, sentiment, view, and belief are often used interchangeably, there are some differences amongst them. An opinion refers to a conclusion open to debate. A view implies a subjective opinion. A belief is a deliberate acceptance and intellectual assent. A sentiment implies a settled opinion reflective of one's feelings [19]. An example of terminologies used for sentiment analysis, taken from Kharde and Sonawane [20], is as follows:

<SENTENCE> = The story of the movie was weak and boring.

<OPINION HOLDER> = <author>

<OBJECT> = <movie>

<FEATURE> = <story>

<OPINION> = <weak><boring>

<POLARITY> = <negative>.

For this project, we are mainly concerned with a type of sentiment analysis called Sentence-Level Polarity Classification. The "Sentence-Level" part of that term refers to the scope of sentiment analysis. Since we are attempting to identify the polarity of a single tweet, we are simply look at a single sentence. Other scopes of sentiment analysis include document-level and aspect-level. "Polarity Classification" refers to a subtask of sentiment analysis. It is the main subtask of sentiment analysis for its simplicity and popularity. It implies that we are identifying whether a

text expresses a positive, negative, or neutral opinion. Other types of sentiment analysis include fine-grained sentiment analysis (same as polarity classification but with two extra classes: very positive and very negative), feeling and emotion detection (identifying anger, happiness, sadness, etc.), and intent analysis (detecting if a person is interested or not interested).

Recalling the four steps of sentiment text classification mentioned in the previous section, we can now give more details and examples for each step. The first step is gathering data. It is estimated that 80% of the world's data is not structure or organized in a predefined manner [21]. This data comes from sources like emails, chats, social media posts, articles, documents, reviews, surveys, etc. All of these could become sources of data for sentiment analysis. During implementation, the text is usually converted into feature vectors for the model to read. This step is referred to as feature extraction. Some examples of features used for feature extraction include words and their frequencies (bag-of-words), part-of-speech tags, opinion words and phrases, position of terms, negation, and syntax. Once we have extracted the features, there are multiple classification algorithms that can be used. Some of the common ones include a Support Vector Machine, a non-probabilistic model that uses a representation of text examples as points in a multidimensional space, maps the points in distinct areas of the space (each distinct area representing a class/sentiment), and draws a hyperplane distinguishing the regions; Naïve Bayes, a family of probabilistic algorithms that use Bayes's Theorem to predict the category of a text; Linear Regression, a famous statistics approach used to predict some value (Y) given a set of features (X); and Deep Learning, a diverse set of algorithms that imitate animal brains by employing artificial neural networks to process data [22]. After training is completed on a model that uses one of the classification algorithms above, it can be used to predict new text that hasn't been labelled.

4.3. Applications of Sentiment Analysis

Sentiment analysis is applied to a diverse set of real-life sectors. Below we discuss some of these sectors and how sentiment analysis can be used in each of them as discussed in detail on www.monkeylearn.com. One of the most popular applications of sentiment analysis is in social media monitoring, and it can allow us to track trends over time, keep up with a competitor, and tune into a specific point in time. In brand monitoring, sentiment analysis can be utilized to understand

how a brand reputation changes over time, identify potential crises, and prioritize actions. When managing voice of customers, it can be used to understand the nuances of customer experience over time. In customer service, it can help route queries to specific customer support teams and prioritize disgruntled customers. In workforce analytics and voice of employee management, it can help discover and address employee concerns. In product analytics, it can keep tabs on opinions about a product, identify what appeals to people. In market research and analysis, it can help quantify otherwise qualitative information and vice versa, tap into new resources of information or fill in gaps where public data is scarce [22].

5. Support Vector Machines

5.1. A Brief Introduction to SVMs

A Support Vector Machine is a classifier whose algorithm tries to find a hyperplane in N-dimensional space that distinctly separates data points according to their classes [23]. The dimension of the hyperplane depends on the number of features, such that the dimension of the hyperplane = dimension of the feature space - 1. For example, in a 2-dimensional feature space in which the data being trained consists of (x, y) points on a graph, the hyperplane is a straight line as shown in Figure 3. Data points falling on either side of the hyperplane are attributed to difference classes. There are infinite possible ways to draw the hyperplane to separate that data. The objective of the SVM algorithm is to maximize the margin between the hyperplane (also known as the decision boundary) and the different classes, more specifically the points in each of the classes that are closest to the hyperplane. These points are called support vectors. Maximizing this margin distance provides some reinforcement so that future data points can be classified with more confidence.

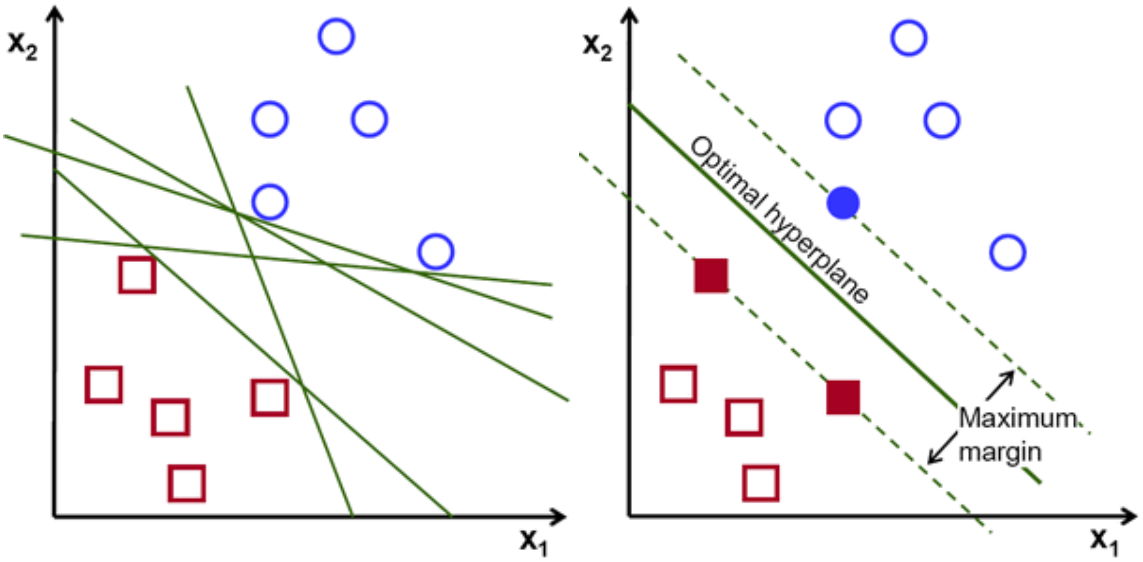

Figure 3: Possible hyperplanes (left) versus hyperplane generated by SVM algorithm (right) [24]

The above example was for linear classification, which means the data points given were linearly separable. In most real-life cases, the data points will not be linearly separable. If the hyperplane that we need to draw is nonlinear then we implement a method called the kernel trick

where we use transformations called kernels [25] that allows us to map the input space into another feature space so that the hyperplane we need to draw now becomes linear. For this project, we are using the LinearSVC classifier provided by Scikit-Learn, which is similar to the regular SVC with a linear kernel parameter. This is because linear SVMs tend to perform well in cases of high dimensional problems, such as our text classification problem [26]. Other popular kernel functions, some of which are shown in Figure 4, are polynomial, exponential, and sigmoid. LinearSVC implements a “one-vs-the-rest” multi-class strategy, which is one of the strategies used to reduce multiclass classification to multiple binary classification problems. It does this by training a single classifier per class, with the samples of that class as positive sample and all other samples as negative samples [27]. There are other multi-class classification strategies, the other widely used one being “one-vs-one” multi-class strategy.

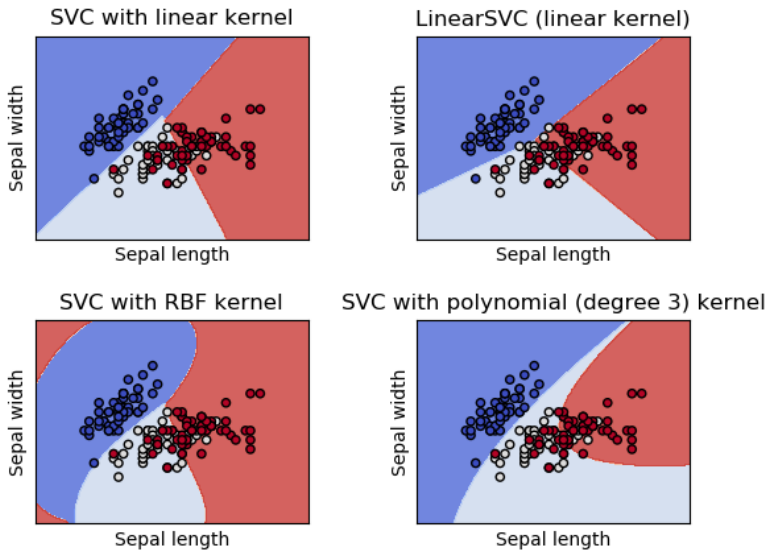

Figure 4: SVM classifiers provided by Scikit-Learn [28]

In the simple example we have been using so far, we were applying hard-margins to the SVM, which means that no data points are allowed to fall into the margins. Therefore, each data point must lie on the correct side of the margin. However, this is only possible when the data points are completely linearly separable. In most real-life cases where the data isn’t linearly separable, we use soft-margins. This means we allows some data to lie in between the margins. However, we still want to maximize the margin between the data points and the hyperplane. Hence, we need to

minimize a loss (error/cost) function. One of the most popular loss functions used on SVMs is the Hinge loss function, which is utilized for maximum-margin (the hyperplane that lies halfway in between the two hyperplanes that separate the support vectors of two classes) classification. The Hinge loss function (Equation 1) punishes misclassification, hence making it extremely useful to determine margins. Another popular loss function is the squared hinge loss function. We also add a regularization parameter to this function to come up with the cost function as shown in Equation 2, which is the average of the loss functions of the entire training set. The Hinge loss function can be seen in a slightly different yet equivalent form inside the summation part of the cost function.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

Equation 1: Hinge loss function

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

Equation 2: Cost function

The purpose of the regularization parameter (termed as C parameter in Scikit-Learn) is to tell the SVM optimization how much you want to avoid misclassifying each training example. Therefore, if C is large, the optimization will choose a hyperplane with a smaller margin, and if C is small, the optimization will choose a hyperplane with a larger margin, even if the hyperplane misclassified more points. Notice that a sufficiently small lambda in the regularization parameter will essentially turn this into a hard-margin classification.

Figure 5: Low regularization value (left) versus high regularization value (right) [24]

Minimization of the cost function is an optimization problem, and there are different ways to solve this problem. The classical approaches involve solving what are referred to as the Primal and Dual problems, which reduce the cost function mentioned above into a quadratic problem. However, Sub-Gradient Descent, which is adapted from Gradient Descent, and Coordinate Descent, which works from the Dual problem, are modern algorithms that have proven to offer significant advantages over the traditional approaches. Sub-Gradient Descent is especially efficient

when training a large data set, and Coordinate Descent is efficient when the dimension of the feature space is high, as is the case for text classification.

Other hyper-parameters of SVMs worth mentioning include tolerance, penalty, and gamma. The tolerance simply refers to when we want to stop training once we believe that our optimization is close enough. The penalty parameter refers to which regularization technique we choose. We have the option of choosing L1 or L2 Regularization, which are also referred to as Lasso Regression and Ridge Regression, respectively [29]. Finally, the gamma parameter defines how far the influence of a single training example reaches, such that a high gamma value only allows for points near the separation line to influence the line while a low gamma value allows far away points to also be considered when calculating the separation line.

Figure 6: Effects of a high gamma value (left) versus a low gamma value (right) [24]

Our choices of hyperparameter values is shown in Table 9 where we use a method called Exhaustive Grid Search to find the most optimal values for our model.

5.2. Pros and Cons of SVMs

There are advantages and disadvantages to choosing SVMs as a classifier. Some of the advantages include the fact that SVMs are effective in high dimensional space and in cases where the number of dimensions is greater than the number of samples. They are memory efficient since they only require the support vectors to adjust hyperparameters and not the whole data set. They also work well when there exist clear margins of separation between classes. On the other hand, the following are some of the disadvantages of SVMs. SVMs requires a long time to train. Therefore, they don't perform well when the training data set is large. They also don't perform well when the data set has too much noise, i.e. when target classes overlap. Lastly, SVMs don't directly provide probability estimates [30].

For this work, we performed text-classification, which is notorious for having large feature vectors. This means we will be using data in high dimensional space and hence will particularly

benefit from SVMs. Furthermore, SVMs only consider the support vectors when drawing hyperplanes for the classes and hence perform just as well on small training data sets. Since we collected and labelled our own training data set and did not have enough time to collect lots of data, we don't have a large training data set. Plus, the dimension of our feature sample is also going to be much greater than the number of samples. Hence, SVMs are the ideal choice of model for us.

5.3. Applications of SVMs

SVMs are mainly used to solve problems that require classification. Here, we mention some common applications. SVMs can be used for face detection, which requires classifying parts of an image as a face or non-face and create a square boundary around the face; text and hypertext categorization, which categorizes texts into different classes; image classification, which assigns an image to a class; bioinformatics, which requires classification of proteins, cancer, and genes; protein fold and remote homology detection, which involves protein remote homology detection; handwriting recognition, which is used to recognize handwritten characters; and generalized predictive control (GPC), which uses SVM-based GPC to control chaotic dynamics with useful parameters [31].

6. Methodology

6.1 Data Set

We used GetOldTweets3[4] to mine tweets from twitter.com. Initially the Twitter Search API was being used. However, Twitter Official API imposes several constraints, which made collecting data difficult. Twitter's Standard Search API is free but the oldest tweets that it can fetch are seven days old. On top of that, Twitter imposes harsh request limits. To get tweets published on prior dates, you need Twitter's Premium or Enterprise Search APIs, which allow users to get tweets up to 30-day old or even the full twitter archive. However, they cost money, and the free versions still impose request limits on the user. GetOldTweets3, an improvement of the original GetOldTweets-python[5] by Jefferson Henrique, is a python 3 library and a corresponding command line utility for accessing old tweets. Although it is not complete, it bypasses some of Twitter Official API's constraints by allowing the user to get tweets older than 7 days which is allowed because tweets in public view can legally be used for academic research. GetOldTweets3 is simple to use and allows the user to specify tweets by username, popularity, query, bound dates, hashtags, language, and location. Two sets of data were gathered: training and analysis data.

6.1.1 Training Data Set Collection

The data set used as a training set for the model was collected from games during the 2016/17, 2017/18, and 2018/19 EPL seasons in which Leicester City either performed well or poorly. Instead of choosing a pre-labelled data set, we chose to select tweets specifically mentioning Leicester City because after training was done, we only planned to run our model on classifying tweets about Leicester City. Therefore, it made sense to train the classifier on data that mentioned Leicester City. The reason we specifically looked for tweets from games in which Leicester City either performed well or poorly was because most tweets mentioning Leicester City (or most other professional sports teams for that matter) during a match are commentaries and considered neutral tweets. By specifically looking for matches in which Leicester performed well,

[4] https://pypi.org/project/GetOldTweets3/
[5] https://github.com/Jefferson-Henrique/GetOldTweets-python

we are almost certain to find positive tweets. Similarly, a poor performance by Leicester City will allow us to find more negative tweets about Leicester City. This was done simply to save time when collecting training data. Since neutral tweets were abundant, there was no need to specifically look for them.

We consider Leicester to have performed well if they managed to defeat tough opponents, beat another team after a run of back-to-back successful games, or beat a team by a large goal difference. Tough opponents in the EPL consist of teams from the group which is nowadays referred to as the “Top Six” (because they usually are the teams that finish in the top six spots at the end of the season). These teams are Arsenal, Chelsea, Liverpool, Manchester City, Manchester United, and Tottenham Hotspur; and they are currently considered the best teams in the EPL due to their success in the past several decades. Back-to-back successful games are ones in which Leicester City managed to beat or tie against their opponents, especially strong opponents, in two or more successive games. We consider good performances with a large goal difference to be a game in which Leicester City defeated their opponent by three or more goals. On the contrary, poor performances by Leicester City were ones in which they lost to a weak opponent, lost to another opponent after a run of back-to-back poor performances, or lost a match by a large margin. These consist of matches in which Leicester City lost to sides that are in the relegation zone, lost another match after having already lost two or more of their previous matches, or lost by three or more goals. We chose tweets from games that described each of the above situations.

Pursa et al. [32] ran experiments to test the effect of data set size on training tweet sentiment classifiers. They discovered that increasing the data set size improves the performance of the classifier, although the increase in performance due to increase in data set size is diminishing. Therefore, we collected as many tweets as we could label in a reasonable amount of time. We collected 6,065 tweets in total for the training set. These tweets were then manually labelled 1 (positive), 0 (neutral), or -1 (negative). Table 1 shows the distribution of the training tweets. During the actual training of the model, we chose to use only a subset of the neutral tweets we collected and labelled, eventually ending up with 1,146 tweets in total for the training set. The

specifications and reasons for this will be explained in the Implementation section (6.4.1) of this paper.

Reason	Date	Home Team	Score	Away Team	Total Tweets	Positive Tweets	Neutral Tweets	Negative Tweets
Big loss	Thu 18 May 2017	Leicester	1-6	Spurs	687	7	502	178
Big win	Sat 17 Sept 2016	Leicester	3-0	Burnley	932	38	891	3
Big win + back to back win	Sat 6 Apr 2019	Huddersfield	1-4	Leicester	414	23	389	2
Back to back loss	Sun 12 Feb 2017	Swansea	2-0	Leicester	1844	10	1561	273
Back to back win	Sat 1 Apr 2017	Leicester	2-0	Stoke	447	52	381	14
Surprising loss	Sat 3 Dec 2016	Sunderland	2-1	Leicester	486	1	417	68
Surprising win	Wed 26 Dec 2018	Leicester	2-1	Man City	1255	249	992	14
Total					6065	380	5133	552

Table 1: Polarity distribution from training data set

The main heuristic used when labelling the data was first to determine exactly at which aspect the sentiment of the tweet was directed towards. If the sentiment in the tweet was not directed towards Leicester City, the tweet was labelled as neutral. Furthermore, if the tweet was a commentary, it was also labelled as neutral. If the tweet was an unbiased question, it was also labelled as neutral. The reason we did not label all questions as neutral was because we believed that there was presence of tweets with questions that we believed expresses emotions. Otherwise, tweets that expressed a positive emotion towards Leicester City were labelled as positive, and tweets that expressed a negative emotion towards Leicester City were labelled as negative. Table 2 includes examples of sentiment classifications.

Positive
● Rooting like crazy for #LeicesterCity. Players stayed for #ChampionsLeague run, but must focus domestically down the stretch. #EPL ● Leicester City with more miracles! ● Love you #leicestercity ● Leicester City won against Chelsea & Man City in a row. Very fine performance! ● Props to Leicester City ● what a win Leicester city 2 Manchester city 1 Foxes never Quit

Neutral
● Leicester city lmfaooooooooo ● LCFC is Leicester City, mate. ● Live Stream : Swansea City v Leicester City #LCFC http://dlvr.it/NM9s6C ● watching Leicester City Football Club ● Kick-off! Join us for live commentary of Swansea City v Leicester City. You can listen on 104.9 FM & DAB, and Freeview 721. ● Thank you, Leicester City.
Negative
● Lol Leicester City should just get relegated already ● Did Leicester City forget how to play football? ● #LeicesterCity in biggggg trouble ● Leicester City are shambolic ● Leicester City are pathetic ha ● Leicester city just suck. I can't believe they won last year and are just trash this year.

Table 2: Examples of sentiment classifications

For each of those games, we collected tweets that mentioned “Leicester City”. GetOldTweets3 only allowed the option of using specific dates to search for tweets, even though it outputted the time as well (in UTC). Hence, we initially extracted tweets for that whole game day. We then got rid of any tweets posted before the kick-off time of the match and two hours after the kick-off time. Therefore, we collected tweets starting from the kick-off time until two hours after the kick-off, which is usually the time when the game ends. This ensured that we collected tweets from the whole match because the two hours after kick-off time would include the first half (45 minutes) + stoppage time (usually 1-4 minutes), half-time (15 minutes), and second half (45 minutes) + stoppage time (usually 1-4 minutes). Since we collect tweets for two hours, the data set will include tweets a couple of minutes after the end of the match, which actually turns out to be vital for our experiment. Since people keep tweeting immediately after the end of a match, it allows us to observe how people on twitter react to the results of the match.

6.1.2. Analysis Data Set Collection

The data set on which we performed our analysis make up tweets posted during Leicester City games in the 2015/16 EPL season. For each of the 38 EPL games that Leicester City played, we collected tweets for analysis in a similar fashion as we did for the training data set. The only games for which there weren’t enough data were Leicester City (3) vs Stoke City (0) on January 23, 2016 and Manchester City (1) vs Leicester City (3) on February 6, 2016. For the game against

Stoke City, GetOldTweets3 did not provide data for the first 30 minutes of the match. As for the Manchester City match, GetOldTweets3 did not provide data for the whole match. Table 3 shows the games for which we collected data. In total, there were 99,316 tweets collected for analysis. To make collecting tweets efficient, we used a Makefile[6] to write all the console commands that collected the tweets and ran it. An example of the console output for the queries used to collect tweets looks like the following:

```
$ GetOldTweets3 --querysearch "Leicester City" --since 2015-09-19 --until 2015-09-20 --lang en
Downloading tweets...
Saved 2467
Done. Output file generated "output_got.csv".
```

Table 3: Console output example for tweet queries

6.1.3. Pre-Preprocessing

GetOldTweets3 outputs the tweets in a csv file, with the following headers: “date”, “username”, “to”, “replies”, “retweets”, “favorites”, “text”, “geo”, “mentions”, “hashtags”, “id”, “permalink”. Once we have manually extracted two hours of tweets from each day as mentioned in the previous section, we perform what we refer to as pre-preprocessing[7], some processing done right before the texts of the tweets are properly preprocessed. This step involves removing columns from the csv that were unnecessary for your research. The columns removed are the following: "to", "replies", "retweets", "favorites", "geo", "mentions", "hashtags", "permalink". Then an empty “polarity” column is added. This is the column that will be filled out when labelling the tweets. This new data would then be saved as a new csv file and would be ready for text preprocessing.

Match week	Date	Home Team	Score	Away Team	Tweets Collected
1	Sat 8 Aug 2015	Leicester	4-2	Sunderland	1226
2	Sat 15 Aug 2015	West Ham	1-2	Leicester	839
3	Sat 22 Aug 2015	Leicester	1-1	Spurs	1011
4	Sat 29 Aug 2015	Bournemouth	1-1	Leicester	372
5	Sun 13 Sept 2015	Leicester	3-2	Aston Villa	2519

[6] Appendix A
[7] Appendix B

6	Sat 19 Sept 2015	Stoke	2-2	Leicester	1052
7	Sat 26 Sept 2015	Leicester	2-5	Arsenal	2172
8	Sat 3 Oct 2015	Norwich	1-2	Leicester	354
9	Sat 17 Oct 2015	Southampton	2-2	Leicester	520
10	Sat 24 Oct 2015	Leicester	1-0	Crystal Palace	497
11	Sat 31 Oct 2015	West Brom	2-3	Leicester	513
12	Sat 7 Nov 2015	Leicester	2-1	Watford	525
13	Sat 21 Nov 2015	Newcastle	0-3	Leicester	1886
14	Sat 28 Nov 2015	Leicester	1-1	Man Utd	4198
15	Sat 5 Dec 2015	Swansea	0-3	Leicester	1637
16	Mon 14 Dec 2015	Leicester	2-1	Chelsea	5891
17	Sat 19 Dec 2015	Everton	2-3	Leicester	2921
18	Sat 26 Dec 2015	Liverpool	1-0	Leicester	2014
19	Tue 29 Dec 2015	Leicester	0-0	Man City	3703
20	Sat 2 Jan 2016	Leicester	0-0	Bournemouth	571
21	Wed 13 Jan 2016	Spurs	0-1	Leicester	1060
22	Sat 16 Jan 2016	Aston Villa	1-1	Leicester	1134
23	Sat 23 Jan 2016	Leicester	3-0	Stoke	880
24	Tue 2 Feb 2016	Leicester	2-0	Liverpool	3723
25	Sat 6 Feb 2016	Man City	1-3	Leicester	0
26	Sun 14 Feb 2016	Arsenal	2-1	Leicester	6846
27	Sat 27 Feb 2016	Leicester	1-0	Norwich	1412
28	Tue 1 Mar 2016	Leicester	2-2	West Brom	2349
29	Sat 5 Mar 2016	Watford	0-1	Leicester	2690
30	Mon 14 Mar 2016	Leicester	1-0	Newcastle	4219
31	Sat 19 Mar 2016	Crystal Palace	0-1	Leicester	2798
32	Sun 3 Apr 2016	Leicester	1-0	Southampton	4611
33	Sun 10 Apr 2016	Sunderland	0-2	Leicester	5288
34	Sun 17 Apr 2016	Leicester	2-2	West Ham	5124
35	Sun 24 Apr 2016	Leicester	4-0	Swansea	5832
36	Sun 1 May 2016	Man Utd	1-1	Leicester	10175

37	Sat 7 May 2016	Leicester	3-1	Everton	5183
38	Sun 15 May 2016	Chelsea	1-1	Leicester	1571
Total					**99316**

Table 4: Results and tweets collected for each Leicester City game during the 2015/16 EPL season[8]

6.2 Preprocessing

Before feature extraction is implemented, the tweets must go through preprocessing. Preprocessing is a vital step for sentiment analysis. Unfortunately, it is also one of the most underestimated and least talked about steps. Texts on social media, especially Twitter, contain noise and irrelevant information that need to be removed or converted before feature vectors could be extracted. Angiani et al. [33] presented work that analyzed and compared different preprocessing steps found in literature. They applied each one of the most known preprocessing filters, independently, to tweets, and they discovered that all techniques, except for using a dictionary, enhanced the performance of the classifier. Some techniques removed useless noise, while others increased the relevance of some concepts, reduced similar terms and expression forms to their most basic meaning. Therefore, we decided to implement as many of these preprocessing steps into this project. We also took inspiration from the works of Ansari et al.[9] and dkakkar[10] for the implementation of the preprocessing.

6.2.1. Tweet-Level Preprocessing

Similar to Angiani et al. [33], we start with basic and cleaning operations. We first convert all the text into lower case. Next, we replace URLs with a “URL” tag, “#hashtag” with “hashtag” (since hashtags might contain relevant information), and “@handle” with “USER_HANDLE”. Furthermore, we strip away spaces, slashes, quotation marks, and apexes from the beginning and end of the tweets. We remove the “RT” from the start of a tweet in case the tweet was a retweet. We then handle emoticons. That is, we replace emoticons with either a “EMO_POS” tag if they

[8] https://www.premierleague.com/results?co=1&se=42&cl=26
[9] https://github.com/abdulfatir/twitter-sentiment-analysis
[10] https://github.com/dkakkar/Twitter-Sentiment-Classifier

display a positive emotion or an “EMO_NEG” tag if they display a negative emotion. Adapted from Ansari et al., Table 4 shows the list of emoticons, their types, and their replacements that we used for this project. Finally, we replace any occurrences of multiple spaces with just a single space. We then move on to word level preprocessing for further preprocessing steps.

6.2.2. Word-Level Preprocessing

For word level preprocessing, we start with removing punctuations from the beginning and end of a word. We convert more than two repetitions of letters to just two letters. We remove dashes and apexes from the words. We replace negative constructs with “not”. Finally, we use stemming on each word. We did not implement the removal of stop words during the preprocessing step because we implement a method called tf-idf transformation (next section) to remove unnecessary words from our tweets.

Emoticons	**Type**	**Replacement**
:), :), :-), (:, (:, (-:, :’)	Smile	EMO_POS
:D, : D, :-D, xD, x-D, XD, X-D	Laugh	EMO_POS
;-), ;), ;-D, ;D, (;, (-;	Wink	EMO_POS
<3, :*	Love	EMO_POS
:(, : (, :-(,):,) :,)-:	Sad	EMO_NEG
:,(, :’(, :”(	Cry	EMO_NEG

Table 5: List of emoticons and their replacements

6.3. Feature Extraction

For this project, we utilize the simple bag-of-words method for feature extraction. Bag-of-words is arguably the most common feature extraction method for sentences and documents. The method considers a vocabulary of known words and a measure of the presence of known words. It is called “bag” of words because any information about the order of the words is discarded and we are only concerned with the presence of a word in a document, in this case a tweet. The intuition for this method comes from the idea that similar tweets will usually contain similar content, e.g. negative tweets will usually contain the same negative words. Table 6 show an example of the use

of bag-of-words, in this case a bag-of-unigrams. We considered unigrams, bigrams, and trigrams with their frequencies for our model. That is because combinations of certain words could give different meanings and sentiments in text.

Tweets	Tweet 1: “This is a tweet.” Tweet 2: “this tweet is also a tweet.”
Bag-of-unigrams	Tweet 1: [a, is, this, tweet] Tweet 2: [a, also, is, this, tweet]
Feature Names	[a, also, is, this, tweet]
Feature Vectors	Tweet 1: [1, 0, 1, 1, 1] Tweet 2: [1, 1, 1, 1, 2]

Table 6: Example of texts, bag-of-words, and feature vectors

There were several Scikit-Learn methods[11] that made feature extraction a simple process. We used the method TfidfVectorizer, which contains more features than just converting the tweets into bags-of-words. TfidfVectorizer is equivalent to using CountVectorizer followed by TfidfTransformer. CountVectorizer converts a collection of text documents (tweets in our case) to a matrix of token counts as shown in Table 6. TfidfTransformer transforms a count matrix to a normalized tf or tf-idf representation. Tf-idf, which is short for term frequency-inverse document frequency, is a numerical statistic method to filter features by weighting and scoring each of the n-grams using the frequency of words in the text [34]. Since our feature vectors could be large from all the tweets we use, which would significantly increase the dimensionality of our vector space, we use tf-idf to extract significant words for each tweet. The feature vectors returned by TfidfVectorizer are fed into the LinearSVC model. The tf, idf, and tf-idf are expressed by the following equations:

$$tf(t) = \frac{Nb_t}{\sum Nb_t} \qquad idf(t) = log(\frac{Tn_d}{Nb_d}) \qquad tfidf(t) = tf(t) * idf(t)$$

Equation 3: tf Equation 4: idf Equation 5: tf-idf

where Nb_t is the number of times a term appears in a tweet, $\sum Nb_t$ is the total number of terms in a tweet, Tn_d is the total number of tweets, and Nb_d is the number of tweets with term t in it.

[11] https://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_extraction.text

6.4. Training

6.4.1. Adjusting Training Data Set Size

Initially, we started training our model using all 6,065 tweets that we collected and labelled. However, we quickly realized (from looking at classification reports and confusion matrix which will be discussed in detail in the next section) that the significantly large number of neutral tweets in comparison to the amount of negative and positive tweets was not allowing the model to classify negative and positive tweets correctly. That meant most tweets were being classified as neutral. Similarly, there were more tweets being labelled with negative polarity than positive polarity, and this could also be because there were more negative tweets labelled than positive tweets. Unfortunately when it comes to imbalanced data sets, SVMs produce suboptimal classification models [35]. To solve these problems, we simply decided to randomly remove neutral and negative tweets from the training data set until we approximately had equal amount of negative, neutral, and positive tweets. Therefore from the original 6,065 tweets mentioned in Section 6.1.1, we ended up using 1,146 tweets: 381 negative, 385 neutral, and 380 positive tweets. Instead of simply reducing the number of neutral and negative tweets, a better method to solve this problem is discussed in the Future Work section (10) below.

6.4.2. K-Fold Cross-Validation

When assessing the predicting ability of a model, it is important to set aside testing data to test a model's classification ability after training. This method is referred to as cross-validation. Generally, we create three separate data sets from the original training data set: a training set, a validation set, and a test set. The training set is used to train the model, the validation set is used adjust hyperparameters while training the data set, and the testing set is used at the end of training to make sure that the model has not over overfit (learned the training set too well and has a high score on validation data, but is unable to generalize to new data) or underfit (hasn't learned from the training set well). Creating the testing data simply requires removing a subset of the training data for later use. Scikit-Learn allows us to do this easily with the train_test_split function, which splits the original training set in a stratified fashion. Generally, 15-25% of the original training data

set is used as testing data. We decided to use 20% of our original training data set as the testing set. Creating the testing data set was simple. However, there are several methods that could be used when creating the validation data set.

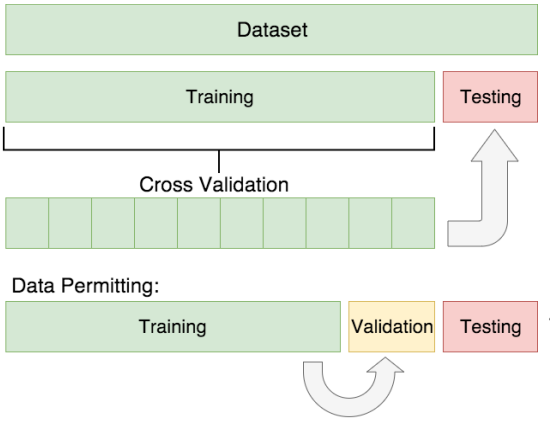

Figure 7: Cross-Validation [36]

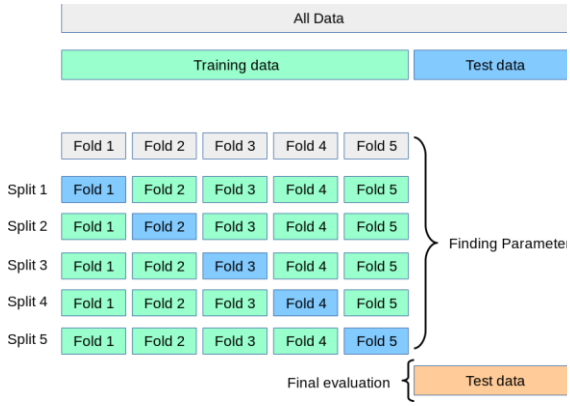

Figure 8: K-Fold Cross-Validation [37]

One of the most popular methods and the method used in this work is referred to as K-Fold Cross-Validation. K-fold divides the rest of the training (after testing data is set aside) into k groups of samples, called folds, of equal sizes. K-1 folds are then used to train the model and the k^{th} fold is used as the validation data set. Each of the k^{th} folds are used as validation sets at least once while the other k-1 are used as training data, and the average of these is taken to finalize the model. This method is computationally expensive, especially depending on the size of your splits. However, it provides a major advantage in problems where the training data set size is small, such as ours because it prevents wasting data compared to when using a fixed, arbitrary validation set. Again, it is recommended to use around 15-25% of the data as validation data. Therefore, we decided to create 5 folds, which means we use 20% of the data as the validation data set during training.

6.4.3. Tuning Hyperparameters using Exhaustive Grid Search

It is possible and recommended to optimize a model's hyperparameters, this is, find the best combination of hyperparameters. We use the Exhaustive Grid Search method, one of the most popular methods for this this, to exhaustively generate candidates from a grid of parameter values

specified. In Scikit-Learn, this is done by specifying the param_grid parameter[12]. The following is a simplified example of the type of grid search we performed for our project:

```
param_grid = [
  {'linear_svc__C': [0.1, 1, 10], 'linear_svc__loss': ['hinge', 'squared_hinge']},
  {'linear_svc__C': [0.1, 1, 10], 'vectorizer__ngram_range': [(1,1), (1,2), (1,3)]}
 ]
```

Table 7: Grid search example

The above param_grid for a LinearSVC model specifies that two grids should be explored: one with C values in [0.1, 1, 10] and a loss function using hinge and squared hinge, and the second one with C values in [0.1, 1, 10] and unigrams, bigrams, and trigrams.

[12] https://scikit-learn.org/stable/modules/grid_search.html

7. Results

Sentiment classification performance is evaluated using four metrics: precision, recall, accuracy, and F1 score [20]. These metrics are defined by the following equations, in which TP, TN, FP, FN refer to the number of true positive instances, the number of true negative instances, the number of false positive instances, and the number of false negative instances, respectively:

$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN} \quad Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

The instances come from the confusion matrix which is defined by the following table:

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

Table 8: Confusion matrix

To summarize these in simpler terms, precision answers the question “from all the data *claiming to be* positive, how much was actually positive?” and recall answers the question “from all the data that *should have been* classified as positive, how much was actually positive?” Accuracy explains the overall accuracy of the model, that is, it answers “from all the data set, how many of it was labelled correctly?” F1 score tries to find a balance between precision and recall. These are the metrics we used to measure our model.

7.1. Measurements

After training, cross-validation, and exhaustive grid search, the following resulted in the best model hyperparameters:

```
TfidfVectorizer params: {'analyzer': 'word', 'binary': False, 'decode_error': 'strict',
'dtype': <class 'numpy.float64'>, 'encoding': 'utf-8', 'input': 'content', 'lowercase': True,
'max_df': 0.9, 'max_features': None, 'min_df': 1, 'ngram_range': (1, 3), 'norm': 'l2',
'preprocessor': None, 'smooth_idf': True, 'stop_words': None, 'strip_accents': None,
'sublinear_tf': False, 'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer': None, 'use_idf':
True, 'vocabulary': None}

LinearSVC params: {'C': 1.0, 'class_weight': None, 'dual': True, 'fit_intercept': True,
'intercept_scaling': 1, 'loss': 'hinge', 'max_iter': 1000, 'multi_class': 'ovr', 'penalty':
'l2', 'random_state': None, 'tol': 0.0001, 'verbose': 0}
```

Table 9: Final model hyperparameter values

Our model outputted the following classification report:

	Precision	Recall	F1-Score	Support
Negative	0.75	0.80	0.77	90
Neutral	0.82	0.73	0.77	70
Positive	0.71	0.73	0.72	70
Micro Average	0.76	0.76	0.76	230
Macro Average	0.76	0.75	0.76	230
Weighted Average	0.76	0.76	0.76	230

Table 10: Final model classification report

We mainly focused on improving the F1-score because improving the F1-score meant that we would be improving both precision and recall scores. Overall, our model had an accuracy and F1-score of 0.76. Our model also had the following confusion matrix and normalized confusion matrix:

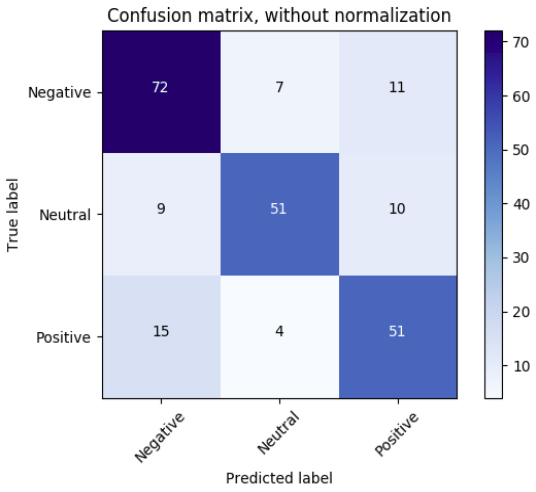

Figure 9: Final model confusion matrix

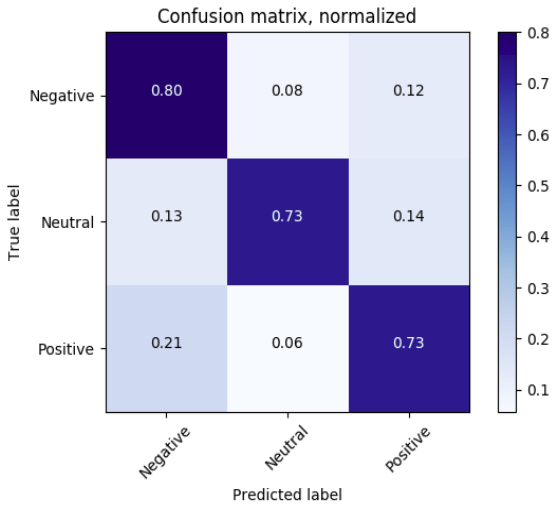

Figure 10: Final model normalized confusion matrix

As shown in the confusion matrices, our model does well when classifying negative tweets, and relatively underperforms when classifying neutral and positive tweets. Furthermore, the misclassified neutral tweets are almost equally divided amongst negative and positive tweets, but most of the misclassified positive tweets are negative.

7.2. Classifying Analysis Data

Once training is done, it is best practice to persist ("save") our model for future use. Therefore, we persisted our model using Python's pickle module. The pickle module is used to implement binary protocols for serializing and de-serializing a Python object structure[13]. Once we pickled the model, we could unpickle it when we needed to classify the 99,316 tweets for analysis. From the 99,316 tweets classified, we got 24,349 positive, 47,336 neutral, and 27,631 negative tweets.

7.3. Possible Noise

The main noise that causes misclassification is one caused during the original labelling of the training data set. Although one's tone changes, the text used when expression feeling towards sporting events is actually quite similar even when the sentiment is different. For example, a tweet saying "COME ON, LEICESTER!" can be classified as having both negative and positive polarities depending on the context. This is one of problems we considered when labelling data and decided to label ambiguous tweets as neutral instead.

Furthermore, we only used a simple bag-of-words approach when extracting features from the tweets. This is perhaps not the most effective feature extraction tool for sporting tweets because there are many instances where a tweet could mention more than one team but have different opinions towards each team. For example, one could mention that they have no opinion towards Leicester City but hate another team in the same tweet. Our model will most likely classify this tweet as negative because of the negative words found in the tweet, even though those negative opinions are not directed towards Leicester City.

[13] https://docs.python.org/3/library/pickle.html

8. Analysis

When performing analysis on the classified data, we were mostly interested in two aspects. The first was how Twitter sentiment on mentions of Leicester City changed throughout the 2015/16 season and the second was how those factors changed within a single game.

8.1. Analyzing Leicester City's 2015/16 Season using Sentiment Analysis

The following graph helps us visualize Table 4:

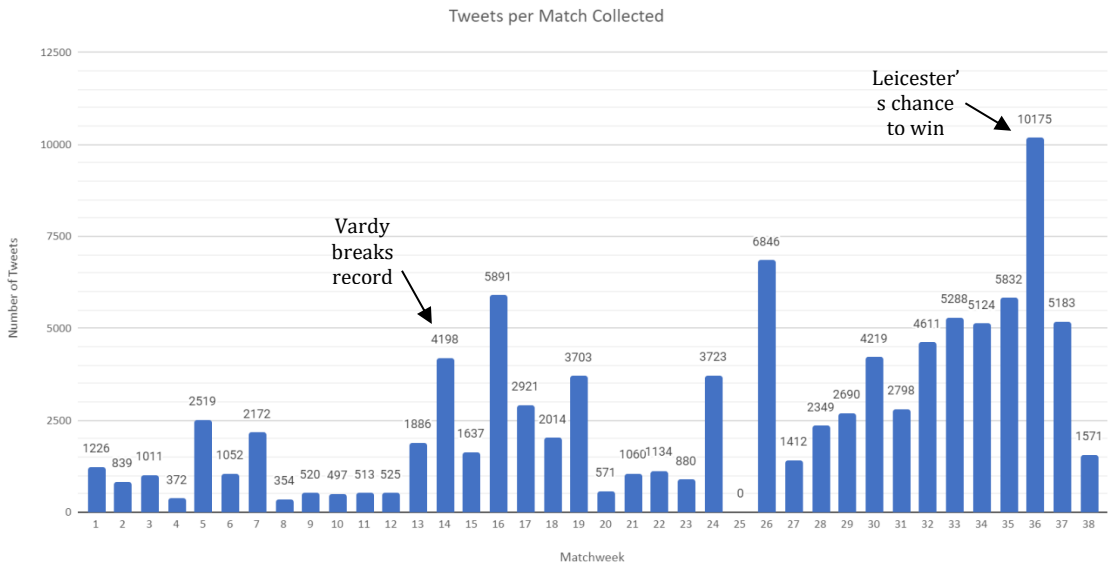

Figure 11: Graph summarizing the number of tweets collected for each matchday (game)

The graph in Figure 11 starts off with small peaks during Matchday 5 and 7. On Matchday 5, Leicester City played an entertaining match in which they came back from 2-0 down to beat Aston Villa 3-2. Matchday 7 saw them lose their first match of the season 5-2 against Arsenal. Afterwards, we notice the number of tweets decrease and level out as Leicester play average against team that aren't in the Big Six. Eventually, we see the number of tweets rise in Matchdays 13 and 14, the games in which Jamie Vardy first equaled then, in the match against Manchester United, broke what was then the current standing record for goals scored in consecutive games in the Premier League. In the next several games, Leicester City played some of the Big Six team, including a 2-1 win over Chelsea on Matchday 16, a 1-0 loss to Liverpool on Matchday 18, and a 0-0 tie against Manchester City in the following week. We see another peak on Matchday 24 when

Leicester get their revenge over Liverpool in a 2-0 win at home, and another large peak during Matchday 26 when they had their hearts crushed by a last-minute Arsenal goal, which saw them lose 2-1. As more and more people continued to talk about Leicester City and their chances for the title, we see a steady rise in the number of tweets per Matchday. The next large peak occurs during Matchday 36, the week Leicester City secured the title. Everyone was talking about this match, as this was the game Leicester could secure their title. Although they missed their chance to beat Manchester United and win the title, Tottenham Hotspur, their title rivals at the time, would eventually tie against Chelsea the next day, securing the title for Leicester City.

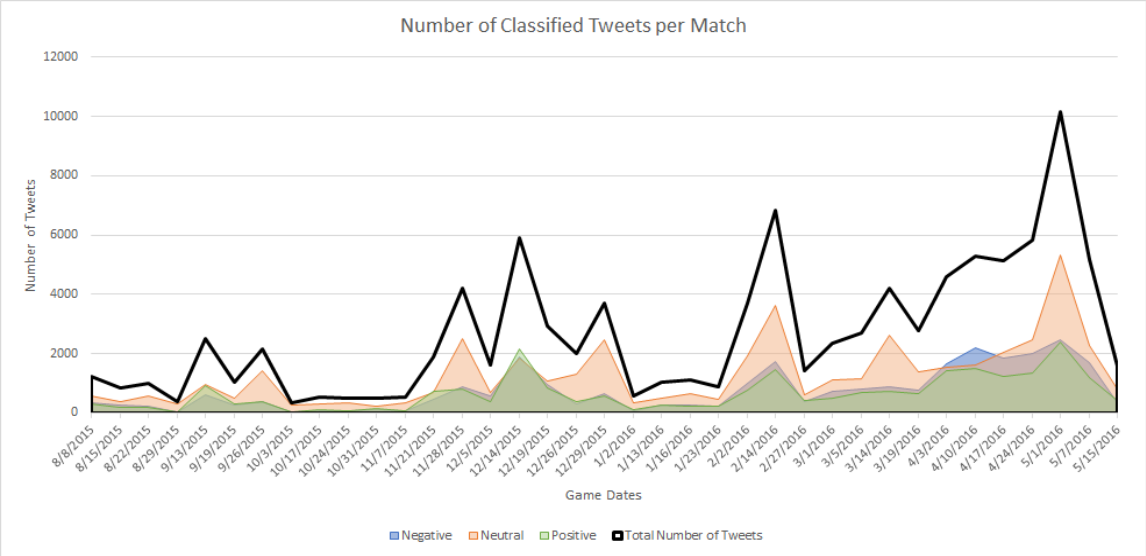

Figure 12: Graph showing the number of tweets for each polarity class per matchday

We ran our model on all those tweets that we collected and Figures 12 and 13 (normalized version of Figure 12) display the polarity distribution that we got for each Matchday. From the non-normalized graph above, we notice that significant events and big games during the season tend to display a large classification of neutral tweets. This shows that these events tend to gather lots of attention from the general public even if the public might not have particular opinions about them, examples being both of Leicester City's games against Manchester United in which Vardy made history and Leicester almost won the title. Another interesting observation we notice from this graph is that some matches generate lots of opinion, while others seem to just get attention. For example, we see a strong expression of opinions during Leicester's match against Chelsea on December 14, 2015. However, we do not see similar results when looking at their match against Manchester

City just two weeks later. A potential explanation for this could be that fans of some teams are more opinionated than others.

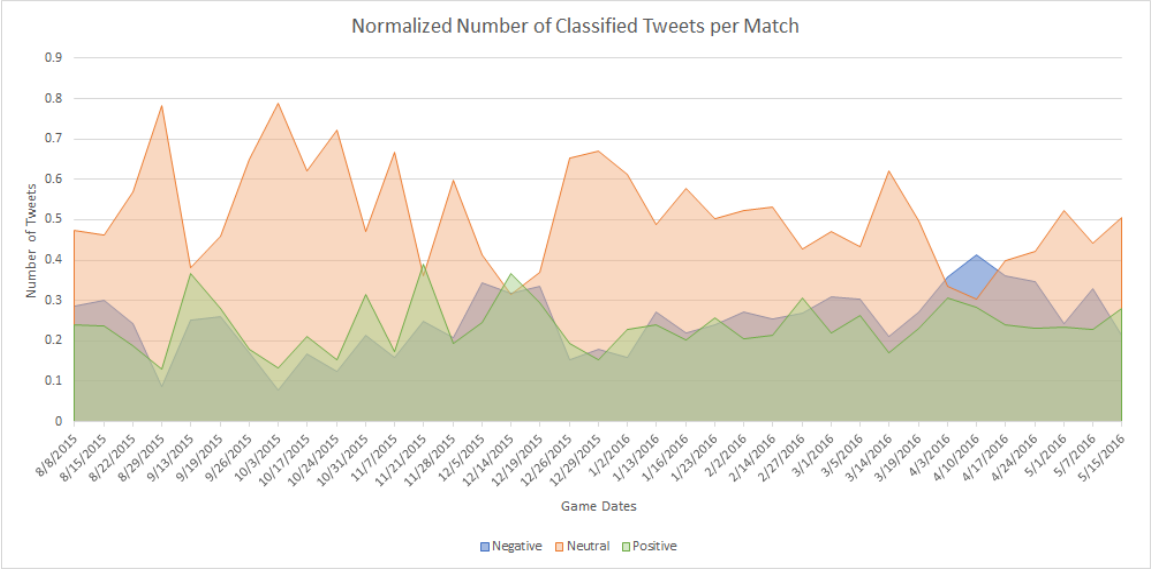

Figure 13: Normalized graph showing the number of tweets for each polarity class per matchday

To get a better understanding of the polarity distribution of the tweets throughout the year, we decided to normalize the above graph to generate the graph found in Figure 13. We did this by taking the number of tweets for one polarity class for one matchday and dividing it by the total number of tweets for that matchday. The most interesting observation from the graph perhaps comes from the fact that during the first half of the season, there were more positive and neutral tweets about Leicester City. However as the season progressed, it seems that people become more opinionated about Leicester City and there was a slight increase in the number of negative tweets about Leicester. An explanation for this could be that fans of the Big Six teams were perhaps becoming frustrated with the sudden realization that their teams were being overtaken by a team that had just survived relegation the season before. It is indeed true that even though Leicester were having a terrific season, none of the Big Six were proud of what they achieved that season. Chelsea, champions of the previous season had a disastrous campaign; Arsenal and Manchester City lacked consistency; Manchester United and Liverpool were in transition; and Tottenham, who were the only team that had truly threatened Leicester, fell behind near the end. Considering how unlikely it was for Leicester to win the title however, they are still considered by many to have staged the greatest upset in sporting history [38].

8.2. Analyzing Manchester United vs Leicester City (Nov 28, 2015)

In this section, we look at the Manchester United versus Leicester City match that took place on November 28, 2015. Although the game ended in a 1-1 draw, it was a celebration for Leicester City as their main striker, Jamie Vardy, broke the record for goals scored in consecutive games in the Premier League. Initially, we notice an increase in the number of tweets sent during halftime, the end of the match, and during the two goals scored in the 24th and 46th minutes as shown in Figure 14. This, for the most part, is self-explanatory. It is normal for fans to focus on the game while it's still being played and tweet during breaks or once the game is over. It is also normal for fans to tweet immediately after a goal. We see similar trends from the results of Gratch et al.'s work [11].

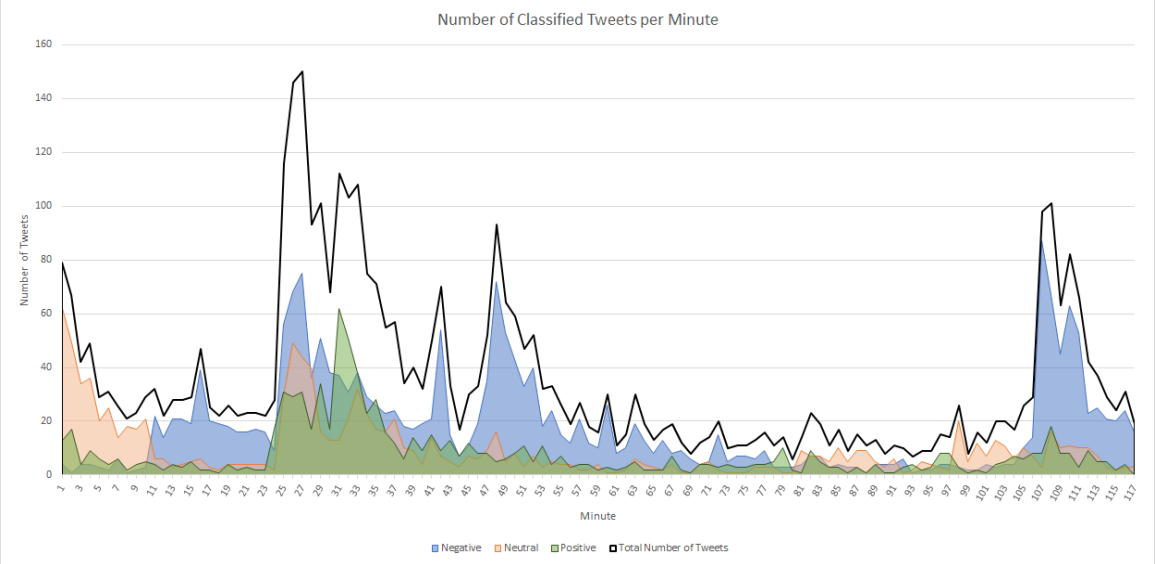

Figure 14: Graph showing the number of tweets for each polarity class during Man Utd vs LCFC

Furthermore, both the normalized and non-normalized graphs below show that the game starts with an abundance of neutral tweets. This is usually normal for soccer games, as most of those tweets are just commentators announcing the start of the game and lineup. Jamie Vardy scored his goal in the 24th minute, and we can observe how this affected the number of tweets sent at that time from Figure 15, which shows a peak of positive tweets at the 24th minute mark.

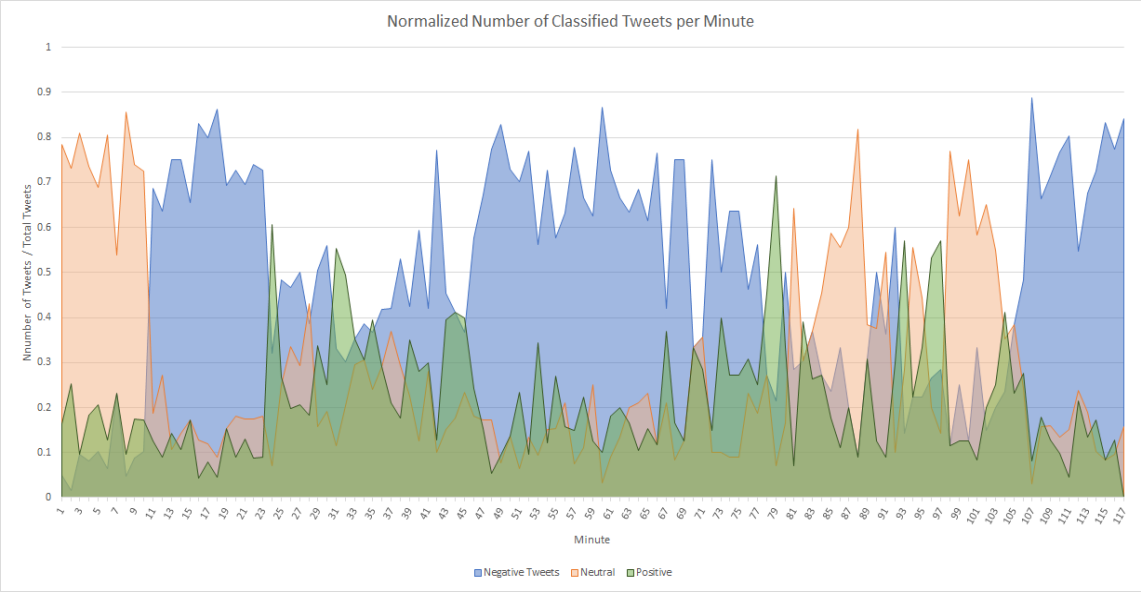

Figure 15: Normalized graph showing the no. of tweets for each polarity class during Man Utd vs LCFC

However, we also notice that the majority of those tweets for that game have negative polarity, especially during halftime and end of the game as we can see in Figure 15. There are several explanations for this. As mentioned before, our model does struggle to classify positive tweets properly. Hence, it might have misclassified some positive tweets as negative (or neutral). Additionally, Gratch et al.'s study [11] revealed that "games with higher tweets per minute also have a higher percentage of negative tweets." Considering this was an important match and hence had more tweets that most of the other matches during the season, it is not surprising for there to be more negative tweets sent during this match. Moreover, Manchester United is arguably considered the biggest soccer club in the world and has a large following, and no fan enjoys watching their team underperform. On top of that, Jamie Vardy broke the record that was held by Ruud van Nistelrooy, who is considered a Manchester United legend. Therefore, it would make sense that most of the reaction, likely from Manchester United fans, was negative. If we had classified tweets mentioning Jamie Vardy instead of Leicester City, we would probably have gotten a different result as there were a lot of tweets congratulating Jamie Vardy for his achievement.

9. Conclusion

We started this paper talking about the rising popularity of microblogging, Leicester City and their amazing season, and other research works done on soccer related events. We gave a brief introduction on machine learning, sentiment analysis, and support vector machines.

For this project, we developed a twitter sentiment prediction model using supervised text classification techniques and a linear SVM. We collected and labelled our own data sets of tweets mentioning Leicester City using GetOldTweets3. After preprocessing our data, we used bag-of-ngrams and vectorization as our feature extraction methods. We improved our model by adjusting the size of our training data set, performing k-fold cross validation during training, and using exhaustive grid search to adjust our hyperparameters. Our model had an F1-score of 0.76.

We used our model to analyze tweets from Leicester's 2015/16 EPL season. We analyzed tweets for all the games in that season, and the main observations from this analysis were that big and important matches generated lots of tweets, especially neutral ones; some games produced more opinionated tweets that others; as the season progressed, the number of tweets mentioning Leicester increased; and there were more positive tweets about Leicester City at the beginning of the season than at the end. We also analyzed tweets for Leicester's game against Manchester United and observed that the game started with a large number of neutral tweets, which is expected for most games; there was a sudden rise in positive tweets when a player had an extraordinary personal achievement; and that fans of Manchester United supported their team and showed negative sentiment when their team underperformed.

10. Future Work

Although GetOldTweets is simple to use, it is limited in how many tweets it returns. Twitter Stream or Search APIs would allow us to extract more data for analysis. It would also be beneficial to perform aspect-level instead of sentence-level sentiment classification on sporting related tweets since people can potentially express different emotions towards different teams in the same tweet. It would also be interesting to perform other sentiment analysis tasks on sporting related tweets such as feeling and emotion detection, which would allow us to identify emotions like anger, happiness, sadness, etc. in fans' tweets. We would also like to use different types of feature extraction tool used such as part-of-speech tagging and dependency parsing, which would allow us to build relationships between words in tweets. Furthermore, we would like to improve the performance of our model by using a larger training data set. This can be done by using a pre-labelled data set or collecting more tweets and using a crowd-sourcing tool to label the tweets.

Furthermore, instead of removing neutral and negative tweets to match the number of positive tweets for training, we should use cost-sensitive classification and assign varying penalties for each class as done by Wasi et al. [39] and adopted from Batuwita and Palade [35]. Finally, we would like to build a program that would allow users to enter a professional team's or individual's name from any sport and be able to see Twitter sentiment trends from any specified dates or times, as allowed by Twitter. This would involve the training of a much larger data set from different sports and integration of Twitter's API to our program.

11. References

[1] D. Sayce, “Number of tweets per day?,” *David Sayce*, 03-Jan-2019. [Online]. Available: https://www.dsayce.com/social-media/tweets-day/.

[2] T. Blog, "Insights into the #WorldCup conversation on Twitter," in *Twitter Blog*, ed, 2014.

[3] J. Percy, “Leicester City season review 2015-16: How did Claudio Ranieri get it so right? ,” *The Telegraph*, 15-May-2016. [Online]. Available: https://www.telegraph.co.uk/football/2016/05/15/leicester-city-season-review-2015-16-how-did-claudio-ranieri-get/.

[4] S. Borden, “The Remarkable Rise of Leicester City,” *The New York Times*, 29-Apr-2016. [Online]. Available: https://www.nytimes.com/2016/05/01/sports/soccer/how-leicester-city-went-right-side-up.html.

[5] P. Carr, “How Leicester City's 5,000-1 odds compare to other long shots,” *ESPN*, 02-May-2016. [Online]. Available: http://www.espn.com/chalk/story/_/id/15447878/putting-leicester-city-5000-1-odds-perspective-other-long-shots-espn-chalk.

[6] *Leicester City manager Claudio Ranieri and captain Wes Morgan lift the Premier League Trophy*. 2016. [Online image]. Available: https://www.premierleague.com/match/12474.

[7] “Leicester City's Riyad Mahrez wins PFA Player of the Year award,” *The Guardian*, 24-Apr-2016. [Online]. Available: https://www.theguardian.com/football/2016/apr/24/leicester-riyad-mahrez-pfa-player-of-the-year.

[8] “Leicester City win Premier League title after Tottenham draw at Chelsea - BBC Sport,” *BBC News*. [Online]. Available: https://www.bbc.com/sport/football/35988673. [Accessed: 28-Apr-2019].

[9] P. Barnaghi, P. Ghaffari, and J. G. Breslin. "Text Analysis and Sentiment Polarity on FIFA World Cup 2014 Tweets". *Conference ACM SIGKDD 2015*, 2015.

[10] Y. Yu and X. Wang. World cup 2014 in the twitter world: A big data analysis of sentiments in us sports fans’ tweets. *Computers in Human Behavior*, 48:392–400, 2015.

[11] G. Lucas, J. Gratch, N. Malandrakis, E. Szablowski, E. Fessler and J. Nichols, "GOAALLL!: Using sentiment in the world cup to explore theories of emotion", *Image and Vision Computing*, vol. 65, pp. 58-65, 2017.

[12] G. Van Oorschot, M. Van Erp and C. Dijkshoorn, "Automatic extraction of soccer game events from twitter", in *Proceedings of the Workshop on Detection, Representation, and Exploitation of Events in the Semantic Web (DeRiVE 2012)*, vol. 902, pp. 21-30, 2012.

[13] S. Jai-Andaloussi, I. El Mourabit, N. Madrane, S. Chaouni and A. Sekkaki, "Soccer events summarization by using sentiment analysis", in *2015 international conference on computational science and computational intelligence (csci)*, 2015, pp. 398-403.

[14] R. Schumaker, A. Jarmoszko and C. Labedz, "Predicting wins and spread in the Premier League using a sentiment analysis of twitter", *Decision Support Systems*, vol. 88, pp. 76-84, 2016.

[15] M. Varone, D. Mayer, and A. Melegari, “What is Machine Learning? A definition,” *Expert System*, 05-Oct-2017. [Online]. Available: https://www.expertsystem.com/machine-learning-definition/.

[16] V. Kurama, “Introduction To Machine Learning,” *Towards Data Science*, 15-Jul-2017. [Online]. Available: https://towardsdatascience.com/introduction-to-machine-learning-db7c668822c4?gi=1f7f33d45e1f.

[17] S. Shukla, “Regression and Classification | Supervised Machine Learning,” *GeeksforGeeks*, 01-Dec-2017. [Online]. Available: https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/.

[18] S. B. Wasi, “Overview of Supervised Sentiment Classification of Tweets”, Image in *2015 Senior Thesis Project Reports*. 2015.

[19] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1-2):1–135, 2008.

[20] Vishal A. Kharde, S.S. Sonawane. Sentiment Analysis of Twitter Data: A Survey of Techniques. *International Journal of Computer Applications (0975 8887)*, vol. 139, no. 11, April. 2016.

[21] C. Schneider, “The biggest data challenges that you might not even know you have,” *IBM*, 16-May-2017. [Online]. Available: https://www.ibm.com/blogs/watson/2016/05/biggest-data-challenges-might-not-even-know/.

[22] “Sentiment Analysis: nearly everything you need to know,” *MonkeyLearn*, 20-Jun-2018. [Online]. Available: https://monkeylearn.com/sentiment-analysis/.

[23] R. Gandhi, “Support Vector Machine - Introduction to Machine Learning Algorithms,” *Towards Data Science*, 07-Jun-2018. [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[24] *Possible hyperplanes*. 2018. [Online image]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[25] S. Patel, “Chapter 2 : SVM (Support Vector Machine) - Theory,” *Medium*, 03-May-2017. [Online]. Available: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72.

[26] T. Joachims, “Text categorization with support vector machines: Learning with many relevant features,” in *European Conference of Machine Learning*. Berlin, Germany: Springer-Verlag, 1998, pp. 137–142.

[27] C. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2016.

[28] *Examples of SVM Classifiers in Scikit-Learn*. 2018. [Online image]. Available: https://scikit-learn.org/stable/modules/svm.html.

[29] A. Nagpal, “L1 and L2 Regularization Methods,” *Towards Data Science*, 13-Oct-2017. [Online]. Available: https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c.

[30] S. Ray, “Understanding Support Vector Machine algorithm from examples (along with code),” *Analytics Vidhya*, 11-Mar-2019. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/.

[31] “Real-Life Applications of SVM (Support Vector Machines),” *DataFlair*, 16-Nov-2018. [Online]. Available: https://data-flair.training/blogs/applications-of-svm/.

[32] J. Prusa, T. M. Khoshgoftaar, and N. Seliya. The effect of dataset size on training tweet sentiment classifier”. In *Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on*, pages 96–102. IEEE, 2015.

[33] G. Angiani, L. Ferrari,T. Fontanini, P. Fornacciari, E. Iotti, F. Magliani,and S. Manicardi. “A Comparison between preprocessing techniques for sentiment analysis in Twitter”. In *2nd International Workshop on Knowledge Discovery on the Web*. KDWeb. 2016.

[34] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up? Sentiment classification using machine learning techniques. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 79–86, 2002.

[35] R. Batuwita and V. Palade, "FSVM-CIL: Fuzzy Support Vector Machines for Class Imbalance Learning", *IEEE Transactions on Fuzzy Systems*, vol. 18, no. 3, pp. 558-571, 2010.

[36] Joseph Nelson, *Visualization of Cross-Validation*. 2018. [Online image]. Available: https://medium.com/@annabiancajones/sentiment-analysis-on-reviews-train-test-split-bootstrapping-cross-validation-word-clouds-4ae65e745f59.

[37] *K-Fold Cross-Validation*. [Online image]. Available: https://scikit-learn.org/stable/modules/cross_validation.html.

[38] J. T., “How Leicester City staged the greatest upset in sporting history,” *The Economist*, 02-May-2016. [Online]. Available: https://www.economist.com/the-economist-explains/2016/05/02/how-leicester-city-staged-the-greatest-upset-in-sporting-history.

[39] S. B. Wasi. 2015 Senior Thesis Project Reports.

12. Appendices

12.1. Appendix A

Makefile

```
default:
	GetOldTweets3 --querysearch "Leicester City" --since 2015-08-08 --until 2015-08-29 --lang
en --output 1.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-08-15 --until 2015-08-16 --lang
en --output 2.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-08-22 --until 2015-08-23 --lang
en --output 3.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-08-29 --until 2015-08-30 --lang
en --output 4.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-09-13 --until 2015-09-14 --lang
en --output 5.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-09-19 --until 2015-09-20 --lang
en --output 6.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-09-26 --until 2015-09-27 --lang
en --output 7.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-10-03 --until 2015-10-04 --lang
en --output 8.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-10-17 --until 2015-10-18 --lang
en --output 9.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-10-24 --until 2015-10-25 --lang
en --output 10.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-10-31 --until 2015-11-01 --lang
en --output 11.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-11-07 --until 2015-11-08 --lang
en --output 12.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-11-21 --until 2015-11-22 --lang
en --output 13.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-11-28 --until 2015-11-29 --lang
en --output 14.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-12-05 --until 2015-12-06 --lang
en --output 15.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-12-14 --until 2015-12-15 --lang
en --output 16.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-12-19 --until 2015-12-20 --lang
en --output 17.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-12-26 --until 2015-12-27 --lang
en --output 18.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2015-12-29 --until 2015-12-30 --lang
en --output 19.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2016-01-02 --until 2016-01-03 --lang
en --output 20.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2016-01-13 --until 2016-01-14 --lang
en --output 21.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2016-01-16 --until 2016-01-17 --lang
en --output 22.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2016-01-23 --until 2016-01-24 --lang
en --output 23.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2016-02-02 --until 2016-02-03 --lang
en --output 24.csv
	GetOldTweets3 --querysearch "Leicester City" --since 2016-02-06 --until 2016-02-07 --lang
```

```
en --output 25.csv
        GetOldTweets3 --querysearch "Leicester City" --since 2016-02-14 --until 2016-02-15 --lang
en --output 26.csv
        GetOldTweets3 --querysearch "Leicester City" --since 2016-02-27 --until 2016-02-28 --lang
en --output 27.csv
        GetOldTweets3 --querysearch "Leicester City" --since 2016-03-01 --until 2016-03-02 --lang
en --output 28.csv
        GetOldTweets3 --querysearch "Leicester City" --since 2016-03-05 --until 2016-03-06 --lang
en --output 29.csv
        GetOldTweets3 --querysearch "Leicester City" --since 2016-03-14 --until 2016-03-15 --lang
en --output 30.csv
        GetOldTweets3 --querysearch "Leicester City" --since 2016-03-19 --until 2016-03-20 --lang
en --output 31.csv
        GetOldTweets3 --querysearch "Leicester City" --since 2016-04-03 --until 2016-04-04 --lang
en --output 32.csv
        GetOldTweets3 --querysearch "Leicester City" --since 2016-04-10 --until 2016-04-11 --lang
en --output 33.csv
        GetOldTweets3 --querysearch "Leicester City" --since 2016-04-17 --until 2016-04-18 --lang
en --output 34.csv
        GetOldTweets3 --querysearch "Leicester City" --since 2016-04-24 --until 2016-04-25 --lang
en --output 35.csv
        GetOldTweets3 --querysearch "Leicester City" --since 2016-05-01 --until 2016-05-02 --lang
en --output 36.csv
        GetOldTweets3 --querysearch "Leicester City" --since 2016-05-07 --until 2016-05-08 --lang
en --output 37.csv
        GetOldTweets3 --querysearch "Leicester City" --since 2016-05-15 --until 2016-05-16 --lang
en --output 38.csv
```

12.2. Appendix B

pre_preprocessor.py

```
"""
Script for pre-preprocessing the raw tweets collected using GetOldTweets3.

Input: csv file generated by GetOldTweets3
Output: ready to be labelled csv file containing only important columns including an empty
"polarity" column
"""

import pandas as pd


# convert csv to a pandas dataframe
tweets_df = pd.read_csv("data-set/raw-data-set/analysis-data-set/analysis-data-set-raw.csv")

# drop unnecessary columns
tweets_df = tweets_df.drop(labels=["to", "replies", "retweets", "favorites", "geo", "mentions",
"hashtags", "permalink"], axis=1)

# add empty "polarity" column
tweets_df["polarity"] = ""

# reorder columns
tweets_df = tweets_df.reindex(columns=["date", "id", "username", "polarity", "text"])
```

```
tweets_df.to_csv("data-set/raw-data-set/analysis-data-set/analysis-data-set-raw-pre.csv",
index=False)
```

12.3 Appendix C

preprocessor.py

```
"""
Script for preprocessing labelled tweets.

Input: csv file of with labelled tweets
Output: preprocessed and labelled tweets
"""

# from nltk.stem.snowball import SnowballStemmer
from nltk.stem.porter import PorterStemmer
import pandas as pd
import re

USE_STEMMER = True


def is_valid_word(word):
    return (re.search(r'^[a-zA-Z][a-z0-9A-Z\._]*$', word) is not None)
    # return word

def preprocess_word(word):
    # remove punctuations
    word = word.strip('\'"?!,.():;')

    # convert more than 2 letter repetitions to 2 letters
    # funnnnny --> funny
    word = re.sub(r'(.)\1+', r'\1\1', word)

    # remove - & '
    word = re.sub(r'(-|\')', '', word)

    # replace negative constructs with "not"
    word = re.sub(r'(cant|dont|isnt|wont|hasnt|arent|aint|never)', 'not', word)

    return word

def handle_emojis(tweet):
    # Smile -- :), : ), :-), (:, ( :, (-:, :')
    tweet = re.sub(r'(:\s?\)|:-\)|\(\s?:|\(-:|:\'\))', ' EMO_POS ', tweet)
    # Laugh -- :D, : D, :-D, xD, x-D, XD, X-D
    tweet = re.sub(r'(:\s?D|:-D|x-?D|X-?D)', ' EMO_POS ', tweet)
    # Love -- <3, :*
    tweet = re.sub(r'(<3|:\*)', ' EMO_POS ', tweet)
    # Wink -- ;-), ;), ;-D, ;D, (;,  (-;
    tweet = re.sub(r'(;-?\)|;-?D|\(-?;)', ' EMO_POS ', tweet)
    # Sad -- :-(, : (, :(, ):, )-:
    tweet = re.sub(r'(:\s?\(|:-\(|\)\s?:|\)-:)', ' EMO_NEG ', tweet)
    # Cry -- :,(, :'(, :"(
```

```
    tweet = re.sub(r'(:,\(|:\'\(|:"\()', ' EMO_NEG ', tweet)
    return tweet

def preprocess_tweet(tweet):
    # print(tweet)

    # convert all text to lowercase
    tweet = tweet.lower()

    # replace URLs with the word URL
    tweet = re.sub(r'((www\.[\S]+)|(https?://[\S]+))', ' URL ', tweet)

    # replace #hashtag with hashtag
    tweet = re.sub(r'#(\S+)', r'\1', tweet)

    # replace @handle with the word USER_MENTION
    tweet = re.sub(r'@[\S]+', 'USER_HANDLE', tweet)

    # strip away space, \, ', and "
    tweet = tweet.strip(' \'"')

    # Remove RT (retweet)
    tweet = re.sub(r'\brt\b', '', tweet)

    # replace emojis with EMO_POS or EMO_NEG
    tweet = handle_emojis(tweet)

    # replace multiple spaces with a single space
    tweet = re.sub(r'\s+', ' ', tweet)

    tweet_as_list = tweet.split()
    preprocessed_tweet = []
    for word in tweet_as_list:
        word = preprocess_word(word)
        if is_valid_word(word):
            if USE_STEMMER:
                # word = str(SnowballStemmer("english").stem(word))
                word = str(PorterStemmer().stem(word))
            preprocessed_tweet.append(word)

    tweet = " ".join(preprocessed_tweet)

    # print(tweet, "\n")
    return tweet

def preprocess_df(tweets_df):
    # iterate through all of the tweet texts in the dataframe and preprocess them
    for index, row in tweets_df.iterrows():
        tweets_df.at[index, "text"] = preprocess_tweet(row["text"])


if __name__ == "__main__":
    # read labelled csv and convert it to a pandas dataframe
        tweets_df = pd.read_csv("testing_preprocessing.csv")

    # conduct preprocessing
        preprocess_df(tweets_df)
```

12.4. Appendix D

main.py

```
"""
Main script that will conduct preprocessing, training, classification, and prediction.
"""


from sklearn.model_selection import train_test_split
import pandas as pd
import preprocessor
import trainer


def split_data(tweets_df):
        """ Splits the data set into training, testing, and validation data sets. """
        tweets = tweets_df["text"].tolist()
        polarities = tweets_df["polarity"].tolist()
        x_train, x_test, y_train, y_test = train_test_split(tweets, polarities, test_size=0.2,
random_state=1)
        return x_train, x_test, y_train, y_test


def main():
        # read labelled csv and convert it to a pandas dataframe
        tweets_df = pd.read_csv("data-set/labelled-data-set/training-data-set/training-data-
set.csv")

        # randomly remove 92.5% of neutral tweets and 31% of negative tweets
        tweets_df = tweets_df.drop(tweets_df.query('polarity == 0').sample(frac=0.925,
random_state=1).index)
        tweets_df = tweets_df.drop(tweets_df.query('polarity == -1').sample(frac=0.31,
random_state=1).index)

        # reindex dataframe and remove old index column
        tweets_df.reset_index(inplace=True)
        tweets_df = tweets_df.drop(columns=["index"])

        # print(tweets_df.shape)
        print(tweets_df.groupby('polarity').size())

        # conduct preprocessing
        preprocessor.preprocess_df(tweets_df)

        # split the dataset into training, testing, and validation data sets
        x_train, x_test, y_train, y_test = split_data(tweets_df)
        print("\nNumber of training data:", len(x_train),"\nNumber of testing data:",
len(x_test))

        # create a classifier and train it using the dataset
        trainer.analyze_model(x_train, y_train, x_test, y_test)

if __name__ == '__main__':
        main()
```

12.5 Appendix E

trainer.py

```
"""
Script that generates the classifier and does the training.
"""


import itertools
import numpy as np
import matplotlib.pyplot as plt
import pickle
from sklearn import svm
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction import stop_words
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV
from sklearn.pipeline import Pipeline


def train(x_train, y_train, x_test, y_test):

        ######################### INIT ###################################
        print("\nINITIALIZING CLASSIFIER...")

        # vectorizer = TfidfVectorizer(ngram_range=(1,2))
        vectorizer = TfidfVectorizer()
        print("vectorizer params:", vectorizer.get_params())

        linear_svc = svm.LinearSVC()
        print("linear svc params", linear_svc.get_params())

        linear_svc_pipeline = Pipeline(steps=[("vectorizer", vectorizer), ("linear_svc",
linear_svc)])


        ################### CROSS VAL and GRID SEARCH ####################
        print("\nPERFORMING GRID SEARCH WITH CROSS VALIDATION...")
        k_fold = KFold(n_splits=20, shuffle=True, random_state=1)
        # k_fold = KFold(n_splits=5, shuffle=True)
        linear_svc_params = [
                {       # Dual optimization
                        "linear_svc__penalty": ["l2"],          # if l1, you can't use hinge
                        "linear_svc__loss": ["hinge", "squared_hinge"],
                        "linear_svc__dual": [True],     # if false, you can't use l2 or hinge
                        # "linear_svc__tol": [1e-4, 1e-5],
                        # "linear_svc__C":[0.001, 0.01, 0.1, 1, 10, 100, 1000],
                        "linear_svc__C":[0.1, 1],
                        # "linear_svc__multi_class": ["ovr", "crammer_singer"],
                        # "vectorizer__stop_words": [None, stop_words.ENGLISH_STOP_WORDS],
                        "vectorizer__ngram_range": [(1,2), (1,3)],
                        "vectorizer__max_df": [0.9, 1.0],
                        # "vectorizer__use_idf": [True, False]
                },
                {       # Primal Optimization
```

```
                        "linear_svc__penalty": ["l1", "l2"],
                        "linear_svc__loss": ["squared_hinge"],
                        "linear_svc__dual": [False],
                        # "linear_svc__tol": [1e-4, 1e-5],
                        # "linear_svc__C":[0.001, 0.01, 0.1, 1, 10, 100, 1000],
                        "linear_svc__C":[0.1, 1],
                        # "linear_svc__multi_class": ["ovr", "crammer_singer"],
                        # "vectorizer__stop_words": [None, stop_words.ENGLISH_STOP_WORDS],
                        "vectorizer__ngram_range": [(1,2), (1,3)],
                        "vectorizer__max_df": [0.9, 1.0],
                        # "vectorizer__use_idf": [True, False]
                }
        ]


        scores = ["precision_micro", "recall_micro", "f1_micro", "accuracy", None]

        for score in scores:
                print("# Tuning hyper-parameters for {0}".format(score))
                print()

                grd = GridSearchCV(linear_svc_pipeline, param_grid=linear_svc_params, cv=k_fold,
scoring=score)
                grd.fit(x_train, y_train)

                print("\nBest score and parameters set found on development set:")
                print("Score:", grd.best_score_, "Params:", grd.best_params_)
                print()

                print("All grid scores on development set:")
                means = grd.cv_results_["mean_test_score"]
                stds = grd.cv_results_["std_test_score"]
                for mean, std, params in zip(means, stds, grd.cv_results_["params"]):
                        print("{0:0.3f} (+/-{1:0.3f}) for {2}".format(mean, std*2, params))
                print()

                print("Detailed classification report:")
                print("The model is trained on the full development set.")
                print("The scores are computed on the full evaluation set.")
                y_true, y_pred = y_test, grd.predict(x_test)
                print(classification_report(y_true, y_pred, target_names=["negative", "neutral",
"positive"]))
                print()

                print("Confusion matrix:")
                print(confusion_matrix(y_true, y_pred))
                print()
                print()

        #create_model(x_train, y_train, x_test, y_test)


def create_model(x_train, y_train, x_test, y_test):
        """ Create a trained model using the best parameters. """

        print("\nCREATING FINAL MODEL...")

        vectorizer = TfidfVectorizer(ngram_range=(1,3), max_df=0.9)
        print("vectorizer params:", vectorizer.get_params())
```

```
        linear_svc = svm.LinearSVC(C=1.0, dual=True, loss="hinge", penalty="l2")
        print("linear svc params", linear_svc.get_params())

        linear_svc_pipeline = Pipeline(steps=[("vectorizer", vectorizer), ("linear_svc",
linear_svc)])

        print("\nTRAINING FINAL MODEL...")
        linear_svc_pipeline.fit(x_train, y_train)

        print("\nPICKLING MODEL...")
        list_pickle = open("final_model/trained_linear_svc.pkl", "wb")
        pickle.dump(linear_svc_pipeline, list_pickle)
        list_pickle.close()

        print("\nUNPICKLING MODEL...")
        list_unpickle = open("final_model/trained_linear_svc.pkl", "rb")
        model = pickle.load(list_unpickle)
        list_unpickle.close()

        print("Detailed classification report for final model:")
        print("The model is trained on the full development set.")
        print("The scores are computed on the full evaluation set.")
        y_true, y_pred = y_test, model.predict(x_test)

        print(model.score(x_test, y_test))
        print(model.get_params)

        print(classification_report(y_true, y_pred, target_names=["negative", "neutral",
"positive"]))
        print()

        print("Confusion matrix for final model:")
        print(confusion_matrix(y_true, y_pred))
        print()
        print()


def analyze_model(x_train, y_train, x_test, y_test):
        print("\nUNPICKLING MODEL...")
        list_unpickle = open("final_model/trained_linear_svc.pkl", "rb")
        model = pickle.load(list_unpickle)
        list_unpickle.close()

        print("Detailed classification report for final model:")
        print("The model is trained on the full development set.")
        print("The scores are computed on the full evaluation set.")
        y_true, y_pred = y_test, model.predict(x_test)

        print(model.score(x_test, y_test))
        print(model.get_params)

        print(classification_report(y_true, y_pred, target_names=["negative", "neutral",
"positive"]))
        print()

        print("Confusion matrix for final model:")
        print(confusion_matrix(y_true, y_pred))
        cnf_matrix = confusion_matrix(y_test, y_pred)
```

```
        np.set_printoptions(precision=2)
        print()
        print()

        plt.figure()
        plot_confusion_matrix(cnf_matrix, classes=['Negative','Neutral', 'Positive'],
                      title='Confusion matrix, normalized')


#Evaluation of Model - Confusion Matrix Plot
def plot_confusion_matrix(cm, classes,
                          normalize=True,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.show()
```

12.6 Appendix F

final-training-output.txt

```
$ python main.py
polarity
-1    381
 0    385
 1    380
dtype: int64

Number of training data: 916
```

```
Number of testing data: 230

INITIALIZING CLASSIFIER...
vectorizer params: {'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype': <class
'numpy.float64'>, 'encoding': 'utf-8', 'input': 'content', 'lowercase': True, 'max_df': 1.0,
'max_features': None, 'min_df': 1, 'ngram_range': (1, 1), 'norm': 'l2', 'preprocessor': None,
'smooth_idf': True, 'stop_words': None, 'strip_accents': None, 'sublinear_tf': False,
'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer': None, 'use_idf': True, 'vocabulary': None}
linear svc params {'C': 1.0, 'class_weight': None, 'dual': True, 'fit_intercept': True,
'intercept_scaling': 1, 'loss': 'squared_hinge', 'max_iter': 1000, 'multi_class': 'ovr', 'penalty':
'l2', 'random_state': None, 'tol': 0.0001, 'verbose': 0}

PERFORMING GRID SEARCH WITH CROSS VALIDATION...
# Tuning hyper-parameters for precision_micro

/home/eyosyaswd/Documents/honors-thesis/honors-env/lib/python3.6/site-
packages/sklearn/model_selection/_search.py:841: DeprecationWarning: The default of the `iid`
parameter will change from True to False in version 0.22 and will be removed in 0.24. This will
change numeric results when test-set sizes are unequal.
  DeprecationWarning)

Best score and parameters set found on development set:
Score: 0.7565502183406113 Params: {'linear_svc__C': 1, 'linear_svc__dual': True,
'linear_svc__loss': 'hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9,
'vectorizer__ngram_range': (1, 3)}

All grid scores on development set:
0.718 (+/-0.146) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 2)}
0.715 (+/-0.141) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 3)}
0.709 (+/-0.152) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 2)}
0.701 (+/-0.135) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 3)}
0.745 (+/-0.163) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.741 (+/-0.143) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.738 (+/-0.152) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.722 (+/-0.134) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.750 (+/-0.139) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 2)}
0.757 (+/-0.134) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 3)}
0.752 (+/-0.143) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 2)}
0.746 (+/-0.131) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 3)}
0.743 (+/-0.144) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.136) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.746 (+/-0.146) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.142) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
```

```
0.513 (+/-0.147) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.479 (+/-0.123) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.536 (+/-0.144) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.491 (+/-0.137) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.745 (+/-0.163) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.741 (+/-0.143) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.738 (+/-0.152) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.722 (+/-0.134) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.694 (+/-0.099) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.688 (+/-0.102) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.689 (+/-0.103) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.681 (+/-0.113) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.743 (+/-0.144) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.136) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.746 (+/-0.146) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.142) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.
              precision    recall  f1-score   support

    negative       0.75      0.80      0.77        90
     neutral       0.82      0.73      0.77        70
    positive       0.71      0.73      0.72        70

   micro avg       0.76      0.76      0.76       230
   macro avg       0.76      0.75      0.76       230
weighted avg       0.76      0.76      0.76       230


Confusion matrix:
[[72  7 11]
 [ 9 51 10]
 [15  4 51]]
```

```
# Tuning hyper-parameters for recall_micro

/home/eyosyaswd/Documents/honors-thesis/honors-env/lib/python3.6/site-
packages/sklearn/model_selection/_search.py:841: DeprecationWarning: The default of the `iid`
parameter will change from True to False in version 0.22 and will be removed in 0.24. This will
change numeric results when test-set sizes are unequal.
  DeprecationWarning)

Best score and parameters set found on development set:
Score:    0.7565502183406113    Params:    {'linear_svc__C':    1,    'linear_svc__dual':    True,
'linear_svc__loss':    'hinge',    'linear_svc__penalty':    'l2',    'vectorizer__max_df':    0.9,
'vectorizer__ngram_range': (1, 3)}

All grid scores on development set:
0.718 (+/-0.146) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 2)}
0.715 (+/-0.141) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 3)}
0.709 (+/-0.152) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 2)}
0.701 (+/-0.135) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 3)}
0.745 (+/-0.163) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.741 (+/-0.143) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.738 (+/-0.152) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.722 (+/-0.134) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.750 (+/-0.139) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 2)}
0.757 (+/-0.134) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 3)}
0.752 (+/-0.143) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 2)}
0.746 (+/-0.131) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 3)}
0.743 (+/-0.144) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.136) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.746 (+/-0.146) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.142) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.513 (+/-0.147) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.478 (+/-0.121) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.536 (+/-0.144) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.491 (+/-0.137) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.745 (+/-0.163) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
```

```
0.741 (+/-0.143) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.738 (+/-0.152) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.722 (+/-0.134) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.693 (+/-0.101) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.687 (+/-0.101) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.692 (+/-0.099) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.681 (+/-0.113) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.743 (+/-0.144) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.136) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.746 (+/-0.146) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.142) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.
              precision    recall  f1-score   support

    negative       0.75      0.80      0.77        90
     neutral       0.82      0.73      0.77        70
    positive       0.71      0.73      0.72        70

   micro avg       0.76      0.76      0.76       230
   macro avg       0.76      0.75      0.76       230
weighted avg       0.76      0.76      0.76       230


Confusion matrix:
[[72  7 11]
 [ 9 51 10]
 [15  4 51]]


# Tuning hyper-parameters for f1_micro

/home/eyosyaswd/Documents/honors-thesis/honors-env/lib/python3.6/site-
packages/sklearn/model_selection/_search.py:841: DeprecationWarning: The default of the `iid`
parameter will change from True to False in version 0.22 and will be removed in 0.24. This will
change numeric results when test-set sizes are unequal.
  DeprecationWarning)

Best score and parameters set found on development set:
Score:    0.7565502183406113    Params:    {'linear_svc__C':    1,    'linear_svc__dual':    True,
'linear_svc__loss':    'hinge',    'linear_svc__penalty':    'l2',    'vectorizer__max_df':    0.9,
'vectorizer__ngram_range': (1, 3)}

All grid scores on development set:
```

```
0.718 (+/-0.146) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 2)}
0.715 (+/-0.141) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 3)}
0.709 (+/-0.152) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 2)}
0.701 (+/-0.135) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 3)}
0.745 (+/-0.163) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.741 (+/-0.143) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.738 (+/-0.152) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.722 (+/-0.134) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.750 (+/-0.139) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 2)}
0.757 (+/-0.134) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 3)}
0.752 (+/-0.143) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 2)}
0.746 (+/-0.131) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 3)}
0.743 (+/-0.144) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.136) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.746 (+/-0.146) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.142) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.513 (+/-0.147) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.478 (+/-0.121) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.536 (+/-0.144) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.491 (+/-0.137) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.745 (+/-0.163) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.741 (+/-0.143) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.738 (+/-0.152) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.722 (+/-0.134) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.693 (+/-0.101) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
```

```
0.688  (+/-0.102)  for  {'linear_svc__C':  1,  'linear_svc__dual':  False,  'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.690  (+/-0.104)  for  {'linear_svc__C':  1,  'linear_svc__dual':  False,  'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.680  (+/-0.111)  for  {'linear_svc__C':  1,  'linear_svc__dual':  False,  'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.743  (+/-0.144)  for  {'linear_svc__C':  1,  'linear_svc__dual':  False,  'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.740  (+/-0.136)  for  {'linear_svc__C':  1,  'linear_svc__dual':  False,  'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.746  (+/-0.146)  for  {'linear_svc__C':  1,  'linear_svc__dual':  False,  'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.740  (+/-0.142)  for  {'linear_svc__C':  1,  'linear_svc__dual':  False,  'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.
              precision    recall  f1-score   support

    negative       0.75      0.80      0.77        90
     neutral       0.82      0.73      0.77        70
    positive       0.71      0.73      0.72        70

   micro avg       0.76      0.76      0.76       230
   macro avg       0.76      0.75      0.76       230
weighted avg       0.76      0.76      0.76       230


Confusion matrix:
[[72  7 11]
 [ 9 51 10]
 [15  4 51]]


# Tuning hyper-parameters for accuracy

/home/eyosyaswd/Documents/honors-thesis/honors-env/lib/python3.6/site-
packages/sklearn/model_selection/_search.py:841: DeprecationWarning: The default of the `iid`
parameter will change from True to False in version 0.22 and will be removed in 0.24. This will
change numeric results when test-set sizes are unequal.
  DeprecationWarning)

Best score and parameters set found on development set:
Score:    0.7565502183406113    Params:    {'linear_svc__C':    1,    'linear_svc__dual':    True,
'linear_svc__loss':    'hinge',    'linear_svc__penalty':    'l2',    'vectorizer__max_df':    0.9,
'vectorizer__ngram_range': (1, 3)}

All grid scores on development set:
0.718 (+/-0.146) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 2)}
0.715 (+/-0.141) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 3)}
0.709 (+/-0.152) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 2)}
0.701 (+/-0.135) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 3)}
0.745  (+/-0.163)  for  {'linear_svc__C':  0.1,  'linear_svc__dual':  True,  'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
```

```
0.741 (+/-0.143) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.738 (+/-0.152) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.722 (+/-0.134) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.750 (+/-0.139) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 2)}
0.757 (+/-0.134) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 3)}
0.752 (+/-0.143) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 2)}
0.746 (+/-0.131) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 3)}
0.743 (+/-0.144) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.136) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.746 (+/-0.146) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.142) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.513 (+/-0.147) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.478 (+/-0.121) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.536 (+/-0.144) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.491 (+/-0.137) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.745 (+/-0.163) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.741 (+/-0.143) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.738 (+/-0.152) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.722 (+/-0.134) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.695 (+/-0.096) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.688 (+/-0.102) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.690 (+/-0.104) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.679 (+/-0.114) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.743 (+/-0.144) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
```

```
0.740 (+/-0.136) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.746 (+/-0.146) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.142) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.
              precision    recall  f1-score   support

    negative       0.75      0.80      0.77        90
     neutral       0.82      0.73      0.77        70
    positive       0.71      0.73      0.72        70

   micro avg       0.76      0.76      0.76       230
   macro avg       0.76      0.75      0.76       230
weighted avg       0.76      0.76      0.76       230


Confusion matrix:
[[72  7 11]
 [ 9 51 10]
 [15  4 51]]


# Tuning hyper-parameters for None

/home/eyosyaswd/Documents/honors-thesis/honors-env/lib/python3.6/site-
packages/sklearn/model_selection/_search.py:841: DeprecationWarning: The default of the `iid`
parameter will change from True to False in version 0.22 and will be removed in 0.24. This will
change numeric results when test-set sizes are unequal.
  DeprecationWarning)

Best score and parameters set found on development set:
Score:    0.7565502183406113    Params:    {'linear_svc__C':    1,    'linear_svc__dual':    True,
'linear_svc__loss':    'hinge',    'linear_svc__penalty':    'l2',    'vectorizer__max_df':    0.9,
'vectorizer__ngram_range': (1, 3)}

All grid scores on development set:
0.718 (+/-0.146) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 2)}
0.715 (+/-0.141) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 3)}
0.709 (+/-0.152) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 2)}
0.701 (+/-0.135) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 3)}
0.745 (+/-0.163) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.741 (+/-0.143) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.738 (+/-0.152) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.722 (+/-0.134) for {'linear_svc__C': 0.1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.750 (+/-0.139) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 2)}
0.757 (+/-0.134) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range': (1, 3)}
```

```
0.752 (+/-0.143) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 2)}
0.746 (+/-0.131) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss': 'hinge',
'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range': (1, 3)}
0.743 (+/-0.144) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.136) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.746 (+/-0.146) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.142) for {'linear_svc__C': 1, 'linear_svc__dual': True, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.513 (+/-0.147) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.479 (+/-0.123) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.536 (+/-0.144) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.491 (+/-0.137) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.745 (+/-0.163) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.741 (+/-0.143) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.738 (+/-0.152) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.722 (+/-0.134) for {'linear_svc__C': 0.1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.695 (+/-0.093) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.684 (+/-0.099) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.690 (+/-0.104) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.679 (+/-0.114) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l1', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}
0.743 (+/-0.144) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.136) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 0.9, 'vectorizer__ngram_range':
(1, 3)}
0.746 (+/-0.146) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 2)}
0.740 (+/-0.142) for {'linear_svc__C': 1, 'linear_svc__dual': False, 'linear_svc__loss':
'squared_hinge', 'linear_svc__penalty': 'l2', 'vectorizer__max_df': 1.0, 'vectorizer__ngram_range':
(1, 3)}

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.
              precision    recall  f1-score   support
```

```
    negative       0.75      0.80      0.77        90
     neutral       0.82      0.73      0.77        70
    positive       0.71      0.73      0.72        70

   micro avg       0.76      0.76      0.76       230
   macro avg       0.76      0.75      0.76       230
weighted avg       0.76      0.76      0.76       230


Confusion matrix:
[[72  7 11]
 [ 9 51 10]
 [15  4 51]]

$ python main.py
 polarity
 -1    381
  0    385
  1    380
 dtype: int64

 Number of training data: 916
 Number of testing data: 230

 CREATING FINAL MODEL...
 vectorizer params: {'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype': <class
'numpy.float64'>, 'encoding': 'utf-8', 'input': 'content', 'lowercase': True, 'max_df': 0.9,
'max_features': None, 'min_df': 1, 'ngram_range': (1, 3), 'norm': 'l2', 'preprocessor': None,
'smooth_idf': True, 'stop_words': None, 'strip_accents': None, 'sublinear_tf': False,
'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer': None, 'use_idf': True, 'vocabulary': None}
 linear svc params {'C': 1.0, 'class_weight': None, 'dual': True, 'fit_intercept': True,
'intercept_scaling': 1, 'loss': 'hinge', 'max_iter': 1000, 'multi_class': 'ovr', 'penalty': 'l2',
'random_state': None, 'tol': 0.0001, 'verbose': 0}

 TRAINING FINAL MODEL...

 PICKLING MODEL...

 UNPICKLING MODEL...
 Detailed classification report for final model:
 The model is trained on the full development set.
 The scores are computed on the full evaluation set.
 0.7565217391304347
 <bound method Pipeline.get_params of Pipeline(memory=None,
      steps=[('vectorizer', TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
         dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
         lowercase=True, max_df=0.9, max_features=None, min_df=1,
         ngram_range=(1, 3), norm='l2', preprocessor=None, smooth_idf=...e', max_iter=1000,
multi_class='ovr',
      penalty='l2', random_state=None, tol=0.0001, verbose=0))])>
              precision    recall  f1-score   support

     negative       0.75      0.80      0.77        90
      neutral       0.82      0.73      0.77        70
     positive       0.71      0.73      0.72        70

    micro avg       0.76      0.76      0.76       230
    macro avg       0.76      0.75      0.76       230
 weighted avg       0.76      0.76      0.76       230


 Confusion matrix for final model:
 [[72  7 11]
  [ 9 51 10]
  [15  4 51]]
```

12.7 Appendix G

classifier.py

```
"""
Script that classifies new data.
"""


import pandas as pd
import pickle
import preprocessor



def classify(model, tweets_df, original_tweets):
        print("\nCLASSIFYING...")
        tweets = tweets_df["text"].tolist()
        tweets_df["polarity"] = model.predict(tweets)
        tweets_df["text"] = original_tweets
        tweets_df.to_csv("data-set/labelled-data-set/analysis-data-set/analysis-data-set-
labelled.csv", index=False)



def unpickle_model():
        print("\nUNPICKLING MODEL...")
        list_unpickle = open("final_model/trained_linear_svc.pkl", "rb")
        model = pickle.load(list_unpickle)
        list_unpickle.close()
        return model



def main():
        # read csv and convert it to a pandas dataframe
        tweets_df = pd.read_csv("data-set/raw-data-set/analysis-data-set/analysis-data-set-raw-
pre.csv")

        original_tweets = tweets_df["text"].tolist()

        # conduct preprocessing
        print("\nPREPROCESSING...")
        preprocessor.preprocess_df(tweets_df)

        # unpickle model
        model = unpickle_model()

        classify(model, tweets_df, original_tweets)


if __name__ == '__main__':
        main()
```

12.8 Appendix H

analysis.py

```
"""
File for analyzing the data we classified.
"""

import pandas as pd


def main():
    # read csv and convert it to a pandas dataframe
    tweets_df = pd.read_csv("data-set/labelled-data-set/analysis-data-set/analysis-data-set-
labelled.csv")
    print(tweets_df.shape)
    print(tweets_df.groupby('polarity').size())


if __name__ == '__main__':
        main()
```